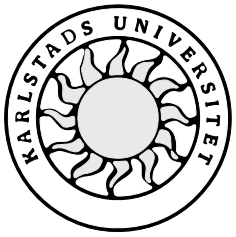


Datavetenskap

Anders Kensby och Johan Strandbergh

Analys och detektering av intrång

i Windows NT

Examensarbete, C-nivå

2002:29

Analys och detektering av intrång

i Windows NT

Anders Kensby och Johan Strandbergh

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Anders Kensby

Johan Strandbergh

Godkänd, 5 juni 2002

Handledare: Stefan Lindskog

Examinator: Stefan Lindskog

Sammanfattning

För att undersöka möjligheterna att upptäcka intrång med hjälp av Window NTs inbyggda loggningssystem, har vi med hjälp av information från tidigare projekt på Karlstads universitet (KaU) reproducerat intrång, designat en databas för intrång och undersökt möjligheterna att jämföra framtagna intrångssignaturer mot de av operativsystemet genererade. Parallellt med detta har vi tagit fram beskrivningar på vanliga säkerhetsbrister, intrångsdetekteringsmetoder och intrångsteknik. Vi kom fram till att inte tillräckligt stor mängd av de program som finns, använder det inbyggda loggsystemet för att det ska vara givande att bygga ett intrångsdetekteringssystem utifrån det. Mycket av vår arbetstid gick åt till att förstå loggstrukturen och att skriva ett program som kan läsa filerna som loggarna lagras i. Utifrån detta kom vi fram till att loggsystemet var onödigt invecklat och inte tillräckligt exakt. Det verkar mer anpassat för att visa läsbara loggar än att ge information lämpad för detektering av händelser.

Analysis and detection of intrusions

in Windows NT

Abstract

The aim of this thesis is to investigate the possibilities to automatically discover intrusions with the help of Windows NTs build-in logging system. We have with information from earlier projects on Karlstads University (KaU) reproduced intrusions, designed a database for intrusions and tried to compare in advance generated intrusion signatures with the ones generated from the operating system. Our conclusion is that not enough of the available programs are using Windows NTs built-in logging system in order for it to be valuable or possible to build a functional Intrusion Detection System (IDS) from it. A large part of our work went into the understanding of the logging structure and to create a program that could read from it. We are also not convinced that enough logs are generated by the operating system itself. The logs are often unnecessary complex and not specific enough. We believe that it is designed to present viewable logs from which conclusions can be drawn from, and not to give information that can be used directly to detect certain events.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Datasäkerhet är i dagsläget en mycket stor marknad. Det finns idag ett oöverskådligt antal företag som verkar inom området och tillsammans med dem kommer också program och verktyg som förbättrar säkerheten på ett eller annat vis. En typ av säkerhetsverktyg är så kallade missbruksdetekteringssystem. Ett sådant varnar en användare eller en system-administratör när vissa fördefinierade händelser, som tyder på missbruk inträffar och kan även utlösa motåtgärder baserat på dessa. Händelserna som verktyget skall reagera på finns lagrade i någon form av databas, vanligtvis är det loggar som jämförs med de som operativsystemet genererar. De frågor vi sätter svar på i denna rapport är: Hur ser möjligheterna ut att skapa ett eget missbruksdetekteringssystem med liknande funktionalitet gällande lagring av händelser, detektering och motåtgärder? Vilka kunskaper, tid och resurser krävs för att utveckla ett detekteringsverktyg?

1.1 Syfte

Under åren 2000 och 2001 utfördes två projektarbeten i kursen tillämpad datasäkerhet. Under kursen skrev studenterna rapporter och deras aktivitet på datorerna loggades. Denna data ses som värdefull, men behöver struktureras och kontrolleras för att sortera ut de felaktigheter som kan förekomma. Det är ett mål hos säkerhetsgruppen att följa upp denna information för att undersöka om den kan användas vidare i liknande arbete och forskning inom datasäkerhet. Datavetenskap vid Karlstads universitet erbjuder därför detta examensarbete. Arbetet består av att praktiskt använda resultaten från projekten som bas för att utveckla ett intrångsdetekteringssystem. Syftet med ett sådant verktyg är att skapa en databas som samlar information från projekten på ett sätt som förhoppningsvis gör den till en resurs för gruppen, då det är svårt att få tag på strukturerad information av det här slaget. Utveckling av ett detekteringssystem kan ge svar på om denna information är lämplig att användas för detektering eller insikt i hur den kan användas samt ger universitetet erfarenhet om vilka möjligheter det finns att detektera intrång och hur det fungerar.

1.2 Mål

Vi har delat upp målen med arbetet i primära och sekundära mål.

1.2.1 Primära mål

- Kontrollera, analysera och strukturera informationen från de datasäkerhetsprojekt som utförts på Karlstads universitet (KaU).
- Ta fram spåren som bildas i loggarna av de intrång som utfördes under projekten och samla dem i en databas tillsammans med information om varje intrång.
- Undersöka möjligheterna att jämföra databasens loggar med de loggfiler som de togs ifrån i syfte att upptäcka intrång.

1.2.2 Sekundära mål

- Utveckla ett verktyg som kan upptäcka de intrång som vi tagit fram signaturer för.

1.3 Förutsättningar

De resurser vi hade till förfogande är följande.

- En laborationsmiljö som efterliknar den som användes i de projektarbeten vi undersöker.
- Dokumentation skrivna av projektgrupperna som beskriver deras aktivitet och resultat.
- Alla loggar som genererades på de datorer som användes under laborationstiden.

1.4 Krav

Detta arbete skall:

- Samla information från datasäkerhetsprojektet.
- Beskriva säkerheten i Windows NT och vanliga säkerhetsbrister däri.
- Ge en uppfattning av på vilka sätt ett system kan attackeras och hur det kan skyddas med hjälp av detektering.
- Förmedla vår erfarenhet av ett försök att utveckla ett missbruksdetekteringsverktyg.

1.5 Rapportens delar

- I kapitel 2 behandlas allmänt säkerheten i Windows NT. Detta för att ge en inblick i hur den ser ut i dagsläget och hur den är uppbyggd.
- I kapitel 3 tar vi upp fem vanliga problem som ofta leder till säkerhetsbrister. Genom att på förhand känna till dessa kan man lättare förhindra att de uppkommer.

- I kapitel 4 ges grundläggande information om intrång och de vanligaste teknikerna för detta tas upp och diskuteras. Vi tar även upp ett praktiskt exempel på hur ett intrång kan gå till.
- I kapitel 5 går vi igenom de rapporter som fanns tillgängliga för oss från tidigare kurser i datasäkerhet på universitetet.
- I kapitel 6 skapar vi en bra grund till en databas som ska lagra information om olika intrång. Den ska även lagra de unika signaturer som skapas vid olika intrång.
- I kapitel 7 går vi igenom de vanligaste typerna av intrångsdetekteringssystem (IDS) som finns på marknaden idag, hur de funkar och varför man ska använda dom.
- I kapitel 8 delar vi med oss av alla de nya lärdomar vi fått under arbetets gång. Detta är allt från vanliga fällor man bör undvika till vad vi anser om de verktyg och system vi arbetat med.
- I kapitel 9 tar vi upp de delar av arbetet som inte blev klara eller som vi annars anser att mer arbete kan läggas ner på.
- I kapitel 10 presenterar vi resultatet av vårt arbete. Vad vi lyckades och vad vi inte lyckades med.

2 Säkerhet i Windows NT

All information i detta kapitel har hämtats från [16].

Säkerhetsmodellen i Windows NT utgörs av fem fundamentala block, som representerar en entitet vilken vanligtvis refereras till som säkerhetsdelsystemet.

Andra säkerhetsblock samverkar med dessa för att kunna erbjuda en komplett säkerhetsarkitektur.

2.1 Security Account Manager (SAM)

Security Account Manager (SAM) har ansvaret att hålla reda på alla lokala användare- och gruppkonton (samt domänkonton hos NT server systemen). SAM databasen innehåller dessutom autentiseringsdata, normalt lösenord, som tillåter den att agera som en självständig enhet för användaridentifikation och verifiering. I denna roll interagerar den mycket med den s.k. ”Local Security Authority” för att validera användarnas anrop.

2.2 Local Security Authority (LSA)

Local Security Authority (LSA) fungerar som en hubb för många av de mer dolda säkerhetsaspekterna. Den tar ansvar för att:

- *Generera accesstokens.* Efter det att inloggningen är avklarad identifieras användaren inte längre med sitt användarnamn utan med dennes säkerhets-id (SID) och associerade accesstokens. LSA tar ansvaret för att bygga en lämplig token.
- *Efterhålla systemets säkerhets- och auditpolicy.* Den s.k. ”Security Reference Monitor” (SRM) frågar LSA för att validera säkerhetskrav. I sin tur frågar LSAn den s.k. ”Security Policy Database” som returnerar ett svar.
- *Logga audit resultat från SRM.* När en audit varning är genererad av SRM är det upp till LSAn att skriva ner varningen till lämplig systemlogg.

2.3 Security Reference Monitor (SRM)

SRM är det primära elementet i säkerhetsstrukturen. En viktig detalj om den är att den arbetar i s.k. ”kernel-mode” och inte i ”user-mode”. SRMs roll är att agera som den primära upprätthållaren för systemets säkerhetsregler.

När en användare vill ha access till ett namngivet objekt är det upp till SRM att undersöka om användaren har rätt till detta. Den erbjuder sedan information om resultatet av prövningen och genererar ett lämpligt audit meddelande som kan loggas av LSAn. Eftersom SRM arbetar i kernel-mode är den ansvarig för både användare- och systemförfrågningar.

2.4 Objekt

Objekt representerar den fundamentala enheten i säkerhetsarkitekturen. All säkerhetsfunktionalitet är byggt runt förutsättningen att Windows NT ska kunna identifiera distinkta element inom systemet; dessa kallas för objekt. Objekt kan inkludera alla typer av resurser som kan bli distinkt identifierade och skyddade. Detta kan inkludera resurser som är uppenbara från en användares gränssnitt såsom filer och skrivare till bakomliggande resurser, såsom trådar och processer.

Mer specifikt kan objekt inkludera följande:

- Fil- och katalogobjekt (När NTs egna filsystem, NTFS, används)
- Tråd- och processobjekt
- Semafor- och händelseobjekt

- Namngivna pipeobjekt
- Portobjekt

Anledningen bakom objektifieringen av resurser är att man ska kunna erbjuda en allmän metod för att kunna påverka dem. En del av denna påverkan är självklart säkerhetsaspekten.

Mer specifikt så tillåter objektiveringen oss att:

- Skapa och explicit ge namn åt givna resurser.
- Skapa accesskontroll över vem som är eller inte är tillåten att använda en given resurs.
- Hålla reda på och logga användare som använder en given resurs.
- Hålla reda på och logga resurser som i sin tur används av den resursen.

2.5 Objekt och ACLs

Som nämnt ovan kan objekt vara vilken namngiven resurs som helst inom systemet. Som man kanske kan förvänta sig är användarnas rättigheter till dessa objekt ytterst beroende på objektet i sig.

Till exempel så är följande rättigheter associerade med filobjekt :

- *Läs* – användare kan läsa filen
- *Skriv* – användare kan skriva till (lägga till eller ändra) filen
- *Ta bort* – användare kan ta bort filen
- *Ändra rättighet* – användare kan ändra rättigheterna på filen
- *Exekvera* – användare kan exekvera (köra) filen
- *Ta över* – användare kan överta filens ägandeskap

Därför har varje användare specifika noll eller flera av ovanstående rättigheter för varje fil. Kombinationen av objekt, objekträttigheter och användarattribut tillåter Windows NT att utföra ett godkännande eller avslag på alla access begäran till objekt.

2.6 Access Control Lists

Varje objekt har två Access Control Lists (ACLs) associerade med sig; en Discretionary ACL, som representerar rättigheterna som kan tilldelas, och en System ACL som sätts av systemets säkerhetspolicy. En discretionary ACL är en lista av de rättigheter som är associerade med objektet och användarna (UserSIDs) [bilaga F] som får använda sig av dessa attribut. Listan av attribut och användare är representerad i en struktur som i Windows NT kallas för en Access Control Entity (ACE). Av detta kommer att en discretionary ACL är en

lista av ACEs. Varje ACE innehåller de attribut och de rättigheter som är associerade med de attributen, representerade antingen i explicit form (dvs. en explicit lista) eller en allmän form (dvs. använder sig av ett generellt attribut såsom skriv fil).

Exempel på ACEs som kan ingå i en discretionary ACL är:

- Användaren Suzanne har full kontroll över katalogen (dvs. läs, skriv, ta bort, ändra rättighet, exekvera, ta över)
- Användaren Sebastian har läs- och exekvera-rättigheter
- Användaren Bosse har läs- och exekvera-rättigheter
- Gruppen Gäster har explicit nekad access

3 Säkerhetsbrister

Datavetenskap vid Karlstads universitet (KaU) har i samarbete med Chalmers tekniska högskola, bedrivit forskning för att hitta och kategorisera brister i operativsystem som orsakar möjligheterna att kringgå säkerheten [1]. Där har man bland annat identifierat fem vanliga typer av säkerhetsproblem.

- Bristande kontroll av inparametrar
- Bristande användning av kryptering
- Svag autentisering
- Osäker bootstrapning
- Felaktig konfiguration

Nedan ger vi en sammanfattning av deras arbete tillsammans med egna erfarenheter för att visa på ett sätt att kategorisera säkerhetsbrister.

3.1 Bristande kontroll av inparametrar

Om programvaran inte utför kontroll av indata till funktioner kan de få data som inte är anpassad till funktionen. Detta kan leda till en så kallad ”buffer overflow” eller helt enkelt att funktionen inte beter sig som väntat. Efter den informationssökning vi bedrivit under arbetet bedömer vi att denna säkerhetsbrist är den som leder till störst antal dataintrång. Det vanligaste sättet att utnyttja dålig kontroll av inparametrar är genom ”buffer overflow” eller ”smashing the stack”, som det populärt kallas. Det finns en mängd program som har blivit utsatta för ”buffer overflow”-angrepp, bland annat Internet Information Server (IIS), men

även systemfunktioner och protokoll. Ansvaret för kontrollen har programmerarna och som utbildade dataingenjörer kunde vi inte låta bli att fördjupa oss i hur ett dataintrång med hjälp av ”buffer overflow” fungerar, se bilaga D.

3.2 Bristande användning av kryptering

Att bristfälligt implementerad kryptering eller ej tillräcklig grad av den för givet ändamål var så pass vanligt gjorde det till en egen kategori av säkerhetsbrister. Kända misstag i implementationen är bland annat algoritmer som inte tar bort ord i klartext från minnet efter att de blivit krypterade eller att krypteringsnycklar finns kvar på hårddisken.

3.3 Svag autentisering

En användare måste bevisa sin identitet innan han ges tillgång till resurser. Vanligen görs detta med ett lösenord, endast känt av användaren och systemet. De exempel på svag autenticering som nämns i [1] gäller autenticeringsprotokoll inom nätverk och är så kallade ”man-in-the-middle”-angrepp.

3.4 Osäker bootstrapning

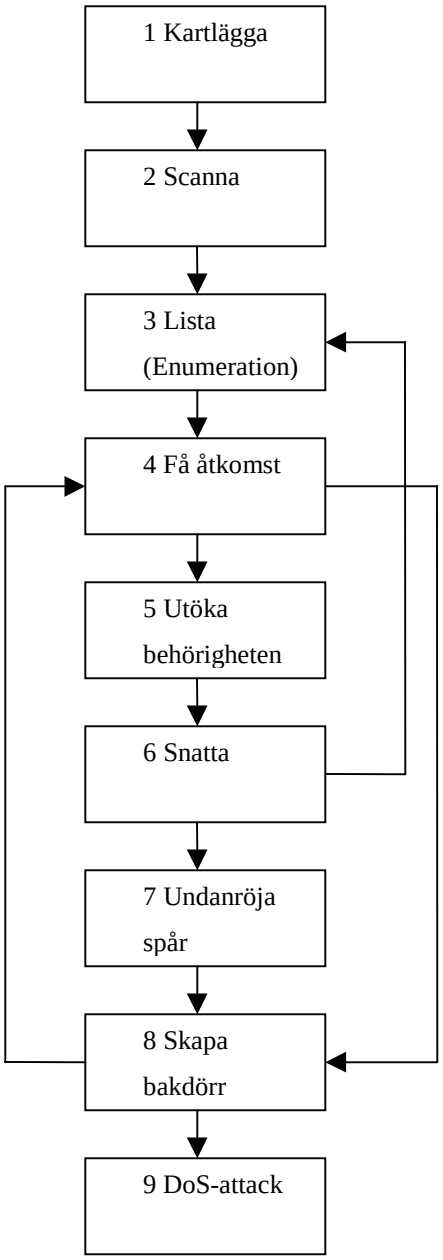
Det finns på de flesta system möjlighet att starta ett datorsystem med ett bootprogram från diskett eller liknande. Detta gör systemet mycket sårbart då det ger möjlighet att initiera ett annat operativsystem, som inte stödjer accesskontroll. Därmed fås tillgång till skyddat material, som till exempel lösenordsfilen.

3.5 Felaktig konfiguration

Med denna kategori menas det arbete som ofta krävs efter att program blivit installerade, för att ställa in en konfiguration som uppfyller kraven på säkerhet. Vanligen har grundinställningen en låg säkerhetsnivå, vilket kan få en ovan administratör att tro han använder ett säkert program eller system, men i själva verket kan programmet eller systemet ha låg eller ingen säkerhet.

4 Intrångsteknik

I detta kapitel tar vi upp de steg som förekommer i de flesta angrepp, vad de innebär, och slutligen ett praktiskt exempel på hur de används för att ta sig in i ett system. Mycket av informationen i detta kapitel har hämtats från [2].



Figur 4-1:Ett hacks anatomi

Nu kommer en kort beskrivning av de nio punkterna som ska försöka klargöra skillnaderna mellan punkterna, vad man vill komma åt med dem och hur det går till väga.

Detta kommer inte vara en steg för steg handbok i hur man bryter sig in i ett system, utan snarare små smakprov på hur man kan gå tillväga för att få tag i information om ett system., eller för administratörer som vill få en insikt i hur angrepp går till för att bättre kunna säkra sina system.

1. **Kartlägga**
    - **Vad:** Samla så mycket information som möjligt om målsystemet och det nätverk det finns i. Detta är en av de stora nycklarna för att kunna fortsätta senare.
    - **Hur:** Öppen källsökning, whois och DNS-zonöverföring.
    - **Med:** Verktygen whois, dig, nslookup. Sökmotorer, USENet och Edgar.
2. **Scanna**
    - **Vad:** Utvärdering av bulkmål och identifiering av tjänster som körs på systemet.
    - **Hur:** Pingsvep, Portscanning och operativsystemsidentifiering.
    - **Med:** Verktygen ping, fping, nmap, superscan, questo och siphon.
3. **Lista**
    - **Vad:** Lista användare och delade resurser som är dåligt skyddade.
    - **Hur:** Lista användarkonton, lista delade filer och identifiera program.
    - **Med:** Null-sessioner, DumpACL, sid2user, OnSite Admin, showmount, NAT, legion, ”banner grabbing” med telnet eller netcat och rpcinfo.
4. **Få åtkomst**
    - **Vad:** Använda den insamlade informationen till att försöka få åtkomst till systemet.
    - **Hur:** Lösenordsavlyssning, Brute Force, hämta lösenordsfilen och buffertspill (s.k. ”buffer overflow” attack).
    - **Med:** Tcpdump, L0pthcrack readsmb, NAT, legion, tftp, pwdump2, ttdb, bind, IIS .HTR/ISM.DLL
5. **Utöka**
    - **Vad:** Försöka få administratörs- eller systemsrättigheter.
    - **Hur:** Lösenordsknäckning, kända intrångsprogram.

- **Med:** john, L0pthcrack, lc_messages, getAdmin, sechole och Bugtraq listor.

6. **Snatta**
    - **Vad:** Använda de nya rättigheterna till att samla på sig mer information och identifiera tekniker för att komma åt tillförlitliga system.
    - **Hur:** Utvärdera förtroendet, söka efter lösenord i klartext.
    - **Med:** rhosts, LSA-hemligheter, användardata, konfigurationsfiler och registret.
7. **Undanröja**
    - **Vad:** Dölja sitt intrång och sin existens på systemet för administratören och andra användare.
    - **Hur:** Rensa loggar, dölja verktyg.
    - **Med:** zap, Event Log GUI, rootkits och fil-streaming.
8. **Skapa bakdörr**
    - **Vad:** Sätta in falluckor och bakdörrar i systemet så angriparen enkelt kan få tillträde till systemet vid ett senare tillfälle.
    - **Hur:** Skapa användarkonton, planera batch-jobb, infektera startfiler, placera ut fjärrkontrollsprogram, installera övervakningsprogram samt ersätta program med trojaner.
    - **Med:** Gruppmedlemsskap, administrators cron, rc, startmappen, registernycklar, netcat, remote.exe, VCN, BO2K och tangentbordsloggare.
9. **DoS-attack**
    - **Vad:** Ofta en sista utväg om angriparen inte lyckas ta kontroll över ett system, se till att ingen annan kan använda det heller.
    - **Hur:** SYN-dränkningar, ICMP-tekniker, Identiska src/dst SYN begäran, Överlappande fragment/offset buggar, OOB och DDoS.
    - **Med:** synk4, pin fog death, smurf, land, latierra, teardrop, bonk, newtear, supermuke.exe och trincoo/TFN/stacheldraht.

4.1 Ett scenario - Du vet på förhand vilket system du ska anfalla

Detta är ofta fallet när man vill bryta sig in i ett specifikt system, kanske för att man vet att säkerheten är extra låg just där, eller kanske är det så att det är brandväggen hos företaget som sparkade dig. I vilket fall så sker valet av målsystem direkt.

Vi kommer i detta exempel ta den primära domänservern som vi använde under arbetet som målsystem. Detta främst då vi själva kontrollerar det och det är därför inte olagligt när vi försöker angripa det.

4.1.1 Kartlägga

Det första som händer är att vi gör en whois-sökning i RIPE-databasen.

RIPE (http://www.ripe.net) är en organisation som har hand om IP-tilldelning i Europa, och erbjuder förutom den tilldelningen att man söker igenom deras egna databas efter adresser de har registrerat. Så vi gör en sökning på 193.10.221.180 och får följande resultat.

```
inetnum:        193.10.220.0 - 193.10.231.255
netname:        SE-KAU
descr:          Karlstad University network
descr:          Karlstad, SWEDEN
country:        SE
admin-c:        GOBO1-RIPE
tech-c:         GOBO1-RIPE
status:         ASSIGNED PI
mnt-by:         SE-KAU-MNT
changed:        goran.bolander@kau.se 19991215
source:         RIPE

route:          193.10.0.0/16
descr:          SUNET C Aggregate
origin:         AS1653
mnt-by:         SUNET-MNT
changed:        bc@sunet.se 19950413
source:         RIPE

person:         Goran Bolander
address:        Karlstad University
address:        S-651 88 Karlstad
address:        Sweden
phone:          +46 54 700 1060
e-mail:         Goran.Bolander@kau.se
nic-hdl:        GOBO1-RIPE
notify:         Goran.Bolander@kau.se
changed:        fredrik@sunet.se 19990422
source:         RIPE
```

Figur 4-2: Resultat av en RIPE-sökning

Whois är även ett trevligt verktyg som finns med i de flesta UNIX-system.

Whois -a kau.se ger oss en del intressant information såsom tillgängliga NS (Name Servers).

Problemet med Whois är att den per default söker en internationell databas, och vi vill gärna ha lite mer nationell information.

Eftersom kau.se är en svensk domän så surfar vi in på NIC-SE (Network Information Centre Sweden AB) som ”tillhandahåller, koordinerar och står för drift av det nationella registret för internetdomännamn under .SE på Internet” enligt NIC-SE som finns att nås på [17].

En sökning i deras databas ger oss mycket mer detaljerad information än den som en manuellt anrop med Whois-verktyget skulle ha gett oss.

```
*domainname.name:                kau.se
*domainname.status:              REGISTRERAT
*domainname.regdate:             1998-09-19

*domainname.dns.data:
kau.se.                    NS    cthns.chalmers.se.
kau.se.                    NS    gungner.dc.kau.se.
kau.se.                    NS    ymer.dc.kau.se.
gungner.dc.kau.se.         A     193.10.220.34
ymer.dc.kau.se.            A     193.10.220.66

*contract_number:                100-515-943
*status:                         UPPRATTAT
*holder:                         Karlstads Universitet
*orgno:                          202100-3120
*contact_info.coaddr:            -
*contact_info.address:           Universitetsgatan 1
*contact_info.zipcode:           651 88
*contact_info.postal_town:       Karlstad
*contact_info.countrycode:       SE
*contact_info.phone_no:          054-83 80 00
*contact_info.fax:               -
*contact_info.contact:           Rolf Johansson
*contact_info.email:             -
```

Figur 4-3: Resultat av en WHOIS-sökning

Efter två enkla sökningar i publika databaser vet vi redan en hel del om nätverket som vårt målsystem befinner sig i.

RIPE sökningen gav oss en översikt över nätverket i sig samt namnet på vem som står som huvudansvarig för säkerheten där, även dennes direktnummer.

NIC-SE var vänliga nog att ge oss allt tänkbar information om de NS som används, tidpunkt för senaste registrering av domänen och till och med den fysiska adressen till platsen där vårt målsystem befinner sig med telefonnummer och kontaktperson.

Om man har planer att genomföra en så kallad ”Social Engineering attack” [9] kan det vara värt mödan att kasta en snabb koll på universitetets hemsida http://www.kau.se.

”Sök Personal” lät som en intressant länk. För att veta lite mer om denne mannen som har huvudansvaret för säkerheten söker jag efter namnet Göran Bolander som jag fick under RIPE sökningen. Här fanns tillgång till ännu mer information om honom, inklusive rumsnummer, tjänst, institution och epost-adress.

Detta skulle kunna användas om vi ville utge oss för att vara denne Göran för att komma över lösenord etc. från människor som kontrollerar system vi vill komma åt.

Vidare kan vi undersöka vad verktyget ”host” ger oss. Host är ett UNIX-verktyg som frågar en Name Server (NS) efter information den känner till.

Använder jag det verktyget direkt på kau.se får jag fram en hel del roliga adresser. Bland dessa finns de två NS som används, deras NS, och en pop3 server cassini.dc.kau.se.

Pop3 servern känns igen på dess typ, MX (mail exchanger).

Sedan utför jag samma fråga igen, men riktar den istället till nätverkets primära DNS-server, gungner.dc.kau.se och får då fram även SMTP-servern munin.dc.kau.se.

Det här är ett exempel på början på den första punkten när man med mycket enkla medel kan börja kartlägga ett nätverk eller företag efter nyttig information.

4.1.2 Scanna

Några enkla saker vi kan börja med vid scanning är att t.ex. utföra en ”ping” mot vårt målsystem för att se om den svarar.

PING 193.10.221.180 (193.10.221.180): 56 data bytes

64 bytes from 193.10.221.180: icmp_seq=0 ttl=127 time=1.4 ms

64 bytes from 193.10.221.180: icmp_seq=1 ttl=127 time=0.7 ms

64 bytes from 193.10.221.180: icmp_seq=2 ttl=127 time=1.6 ms

Datorn svarade direkt på vår ping. Sättet en dator svarar på kan ofta användas för att identifiera operativsystemet som körs, detta sker ofta med så kallad aktiv stackidentifiering.

Eftersom datorn var så snäll och svarade oss direkt kanske vi har sådan tur att den inte är skyddad av en brandvägg eller dylikt. Vi försöker i alla fall härnäst att utföra en s.k. portscanning av datorn.

En portscanning av en dator försöker ansluta till alla TCP- och UDP-portar för att bestämma vilka tjänster som körs eller lyssnar. För detta ändamål kommer vi att använda oss av UNIX-verktyget nmap.

Samtidigt som vi scannar datorn kommer vi att försöka bestämma vilket operativsystem som användes genom så kallad aktiv stackidentifiering.

Tabell 4-1 : Följande portar svarade på vårt pingsvep

Port	**State**	**Protocol**	**Service**
21	open	tcp	ftp
25	open	tcp	smtp
80	open	tcp	http
135	filtered	tcp	loc-srv
136	filtered	tcp	profile
137	filtered	tcp	netbios-ns
138	filtered	tcp	netbios-dgm
139	filtered	tcp	netbios-ssn
161	filtered	tcp	snmp
162	filtered	tcp	snmptrap
443	open	tcp	https
465	open	tcp	smtps
512	filtered	tcp	exec
513	filtered	tcp	login
514	filtered	tcp	shell
515	filtered	tcp	printer
540	filtered	tcp	uucp
1025	filtered	tcp	listen
1029	open	tcp	unknown
1032	open	tcp	iad3
1036	open	tcp	unknown
1243	filtered	tcp	unknown
2000	filtered	tcp	callbook
2001	filtered	tcp	dc
2049	filtered	tcp	nfs
5000	filtered	tcp	fics
9081	open	tcp	unknown
12345	filtered	tcp	NetBus

nmap gissar på någon av följande operativsystem: NT4, Windows 98 eller Windows 95.

Vid en första anblick liknar det ovan nog mest grekiska. Men vad vi nu fått tag på är en lista över alla portar som vårt målsystem svarar på, samt vilken sorts service som ligger bakom och lyssnar.

Vi tänkte nu koncentrera oss på portarna 21, 25 och 80. Detta därför att de representerar några av de vanligaste tjänsterna: ftp-, smtp- och www-server.

Dessa vill vi ha lite mer information om.

Så för att få detta använder vi oss av den populärt kallade ”TCP/IP swiss army knife”, eller netcat. Det är helt enkelt ett program som låter oss göra en TCP-uppkoppling i så kallad raw-mode [5].

Vad vi gör är att helt enkelt försöka koppla upp oss mot de portarna vi var intresserade av.

Mycket riktigt så är målsystemet så snällt att den svarar på portarna och är vänlig nog att presentera sig för oss. Nedan kan svaren ses.

Tabell 4-2: Svar på TCP-uppkoppling i rawmod med netcat

Port	**Service**	**Svar**
21	ftp	Microsoft FTP Service (Version 4.0)
25	Smtp	Microsoft SMTP MAIL ready at Wed, 1 May 2002 18:06:00 +0200 Version: 5.5.1877.197.19 220 ESMTP spoken here
80	http	Microsoft-IIS/4.0

Nu börjar det likna något.

Det är definitivt ett system som körs på Microsoft mjukvara. Vänliga som de är får vi veta att IIS4, Internet Information Server 4, är installerat på datorn.

Vi får även versionsnumret på SMTP-servern.

4.1.3 Få åtkomst

Redan här kan vi faktiskt börja leta svagheter i systemet. IIS t.ex. är känt för att ha säkerhetsproblem, så det är en bra plats att börja.

Men först en liten resumé över vad vi fått fram än så länge.

Systemet körs på Microsoft-mjukvara. Eftersom IIS4 körs så är NT4 det mest troliga valet av dem som gissades av vår portscanner. Den senaste service packet är 6.0, så det är också ett

antagande vi gör. Vi väljer fortsättningsvis att koncentrera våra attacker mot IIS4, då det är känt för att ha många problem.

Vad vi nu letar efter är alltså sårbarheter i IIS4 som körs på NT4 Service Pack 6.

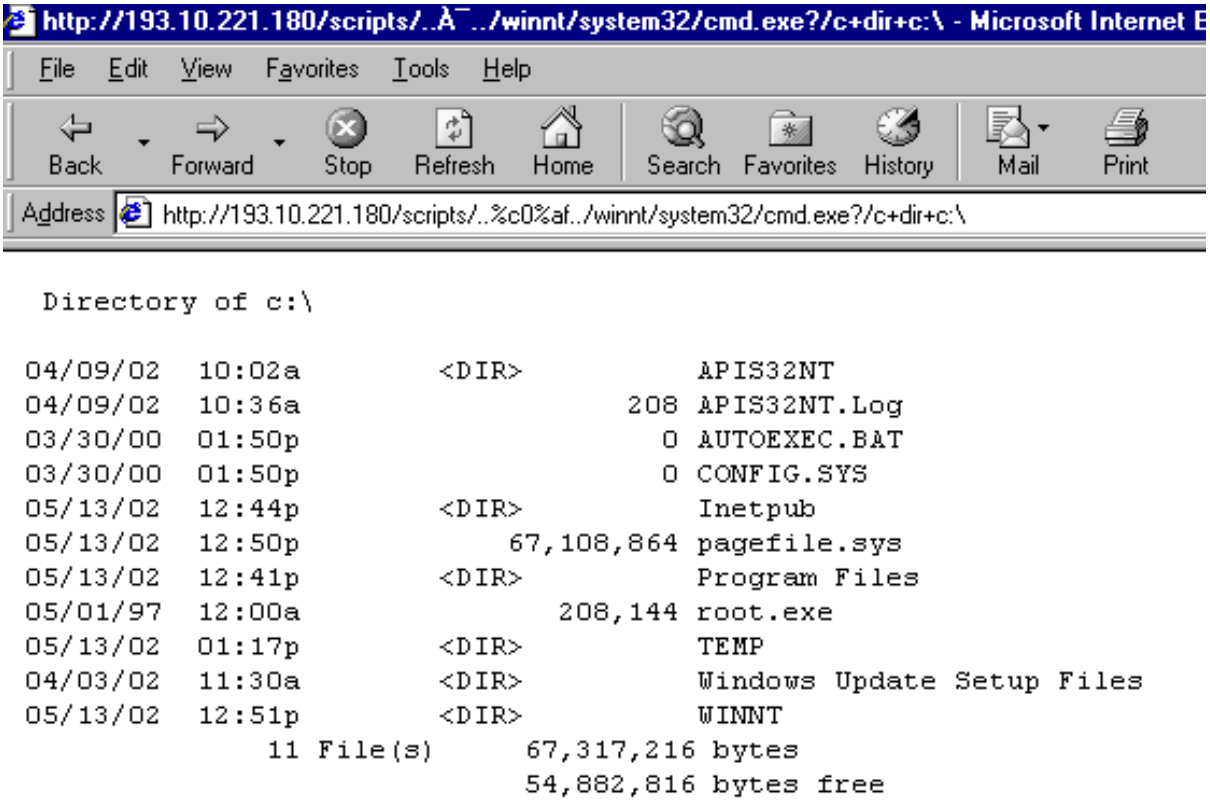
Microsofts egen hemsida är ofta en bra plats att börja då de brukar lägga ut mycket information om buggar i sin programvara i samband med fixar för dem. Med lite tur hittar man ibland något som inte är fixat.

Man kan sedan fortsätta sina sökningar i diverse databaser och nyhetsgrupper på Internet för att hitta något. Vi hittade faktiskt en sårbarhet för just Service Pack 6 och IIS4. Det tog inte speciellt lång tid.

Det finns en så kallad ”buffer overflow” se bilaga D, inuti IISens parsingsmekanism som kan utnyttjas för att få fjärranslutning till datorn med SYSTEM-access. Denna bugg existerar för precis vårt målsystem, NT4 med Service Pack 6 och IIS4. Det är egentligen en lokal bugg, men den kan användas remote, och det är vad vi tänker göra.

Det är en lokal sårbarhet eftersom man måste skapa en illasinnad .asp-fil som när den parsas av IIS kommer orsaka att inetinfo.exe råkar ut för denna ”buffer overflow” och p.g.a. detta kommer låta oss ta kontroll över den lokala servern som SYSTEM.

Hur ska man nu kunna trigga denna brist ”remote”? Jo, vi använder oss av en annan känd brist i ISS som kallas för ”Microsoft IIS Unicode Exploit” [6] samt en .asp-overflow. Nedan kommer ett exempel på just denna brist som låter oss fjärr-lista mappar och dess innehåll på servern.



Figur 4-4:Exempel på Unicode buggen

IIS Unicode exploiten tillåter oss snabbt förklarat att remote exekvera kommandon mot IIS som IUSR_MACHINE. Och eftersom vi kan exekvera kommandon genom cmd.exe kan vi försöka få webbservern att kontakta en utanliggande FTP-server och hämta en fil som sedan ska exekveras som IUSR_MACHINE.

Den ”exploit” vi fann fungerar så att den med hjälp av Unicode buggen få upp vår egenskapade .asp-fil till webservern som där ska hämtas. Detta kommer leda till att inetinfo.exe råkar ut för en ”buffer overflow” och kommer låta oss binda cmd.exe till en av oss vald port där den kommer fungera som en telnet-server. Och eftersom det kommer att vara SYSTEM som startar denna cmd.exe session kommer vi kunna logga in med SYSTEM rättigheter.

Detta är en ganska långsökt ”exploit”, som inte kommer att fungera på ett system som har säkrat upp sig lite. Det visade sig att vårt målsystem inte var speciellt bra säkrat.

Dock är det något som händer lite för ofta. Att människor installerar och kör tjänster som i sig är mycket fina och trevliga, men utan att ha nog med kunskap för att kunna använda dem säkert. Detta är en av de stora anledningarna till att följande ”exploit” fungerar så pass bra den ändå gör.

Vi kompilerar i alla fall programmet som ska utnyttja denna svaghet. När detta är gjort hoppar vi ut till konsolen och startar det.

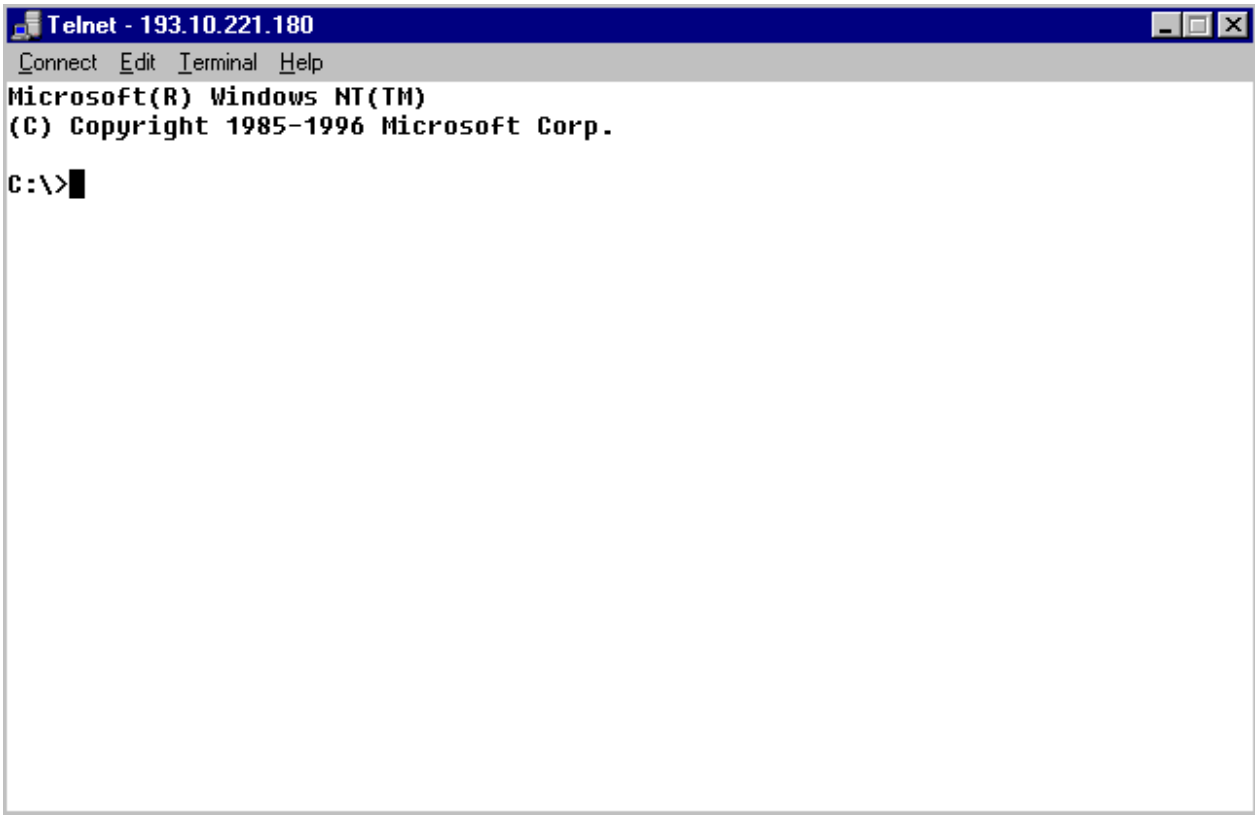
Detta kommer att försöka angripa vårt målsystem på port 80 (www) och öppna en Telnet session åt oss på port 5000. Följande ”flashar” förbi på kommandoraden:

```
Attempting to find an executable directory...
Trying directory [scripts]
Executable directory found. [scripts]
Path to executable directory is [C:\Inetpub\scripts]
Moving cmd.exe from winnt\system32 to C:\Inetpub\scripts.
Successfully moved cmd.exe to C:\Inetpub\scripts\XXX.exe
Sending the exploit...
Exploit sent! Now telnet to 193.10.221.180 on port 5000 and
you should get a cmd prompt.
```

Ok.

Vi testar helt enkelt.

C:\tmp\telnet 193.10.221.180 5000



Figur 4-5: Telnet prompt med SYSTEM rättigheter

Mycket riktigt, vi lyckas få kontakt med vårt målsystem genom Telnet och står nu vid en blinkande prompt med SYSTEM rättigheter. Vad ska vi göra härnäst?

Detta är ett exempel på hur enkelt det egentligen kan vara att bryta sig in i ett system genom enkla medel, som finns tillgängliga för alla med Internetaccess. Genom att sedan fortsätta med resterande punkter kan en ondsint människa göra nästan vad som helst med ett dåligt bevakat system.

5 Datainsamling

För insamling av intrångsdata utnyttjades tidigare arbeten inom området datasäkerhet utförda på KaU tillsammans med Internet.

5.1 Projektet i tillämpad datasäkerhet

Det har utförts två projekt med uppgiften att praktiskt undersöka säkerheten i NT. De utfördes av studenter i kursen tillämpad datasäkerhet år 2000 och 2001.

5.1.1 Beskrivning

Tillvägagångssättet har varit att utan handledning hitta brister i säkerheten och använda dessa för att bryta sig in i målsystemet. De regler som fanns var bland annat att endast en bestämd laborationssal fick användas och tidsperioden var fyra veckor, samt att inte utnyttja rättigheter eller lösenord man kom över på ett oetiskt sätt. En handledare fanns för tekniska eller praktiska frågor men ingen vägledning gavs för att lösa uppgiften. Bilaga A innehåller laborationsspecifikationen för projekten.

5.1.2 Målmiljön

En laborationssal med ett tiotal klientdatorer exklusive en primär och en sekundär NT-server tilldelades uppgiften. För att få en realistisk miljö skedde försöken utan övriga studenters vetskap och en normal administratörsaktivitet uppehölls. Studenterna uppmanades att inte berätta om aktiviteten för att inte påverka andra användare att vara onormalt försiktiga eller på annat sätt skydda sig, detta för att upprätthålla en nivå av realism. På klientdatorerna fanns NT version 4 med service pack 3 respektive service pack 6 installerade, den förstnämnda användes år 2000. På servern kördes Internet Information Server (IIS) version 4.0 i operativsystemet Windows NT. Tjänster som användes var html, ftp och smtp. Se bilaga D för en överblick över IIS 4.0.

5.1.3 Rapportering

Tre typer av rapporter skulle tas fram av studenterna (angriparna), bakgrunds-, aktivitets- och slutrapport, se bilaga B.

I bakgrundsrapporten beskriver deltagarna sina kunskaper och tidigare erfarenheter av datasäkerhet. Detta dokument togs fram för att kunna utvärdera hur mycket tid en person med en viss kunskapsnivå behöver för att göra ett visst intrång och hur kurvan av lyckade intrång mot tiden ser ut för grupper med olika grad av kunskap. Se bilaga B.1.

Aktivitetsrapporterna beskriver samtliga aktiviteter som har utförts under projektet, alltså en rapport per aktivitet. Här ska tid som lagts ner på informationssökning och allmänt funderande finnas, tillsammans med en förklaring av vad som hittades. Dessutom ska alla intrångsförsök finnas beskrivna med bland annat plats och tidpunkt. Dessa rapporter skulle lämnas in kontinuerligt till handledare. Se bilaga B.2.

I slutrapporten sammanfattar varje grupp sitt arbete. Här beskrivs alla lyckade intrång som gruppen utförde under laborationen, hur tiden har disponerats, vilka metoder som använts och vad de kom fram till angående säkerheten i Windows NT. Se bilaga B.3.

Ett mål med utformningen av rapporterna är att man med bakgrundsrapporten och aktivitetsrapporten skall kunna räkna ut hur stor arbetsinsats det krävs att bryta sig in i systemet.

5.2 Verifiering av intrång

Under det första försöket deltog 30 grupper varav 25 lämnade in slutrapporter. Resultaten av grupperna varierar stort då vissa grupper har flera lyckade intrång medan andra inte lyckats alls. För att verifiera deras påstådda resultat har vi med hjälp av aktivitetsrapporten och loggfilerna från laborationsmiljön kunnat kontrollera deras beskrivning av händelseförloppet.

Aktivitetsrapporten är konstruerad för att detta ska vara möjligt och innehåller därför bland annat information som anger vilken arbetsstation som användes och vid vilken tidpunkt. Samtliga grupper verkar dock inte varit medvetna om anledningen och vikten av att ange dessa då flera angett ett tidsspann som är stort jämfört med den enskilda sekvensen av händelser rapporten beskriver. Vi nöjde oss därför att verifiera en delmängd av intrången.

5.2.1 Lyckade intrång

Nedan följer en beskrivning på en delmängd av de lyckade intrången. Kontroller vi gjort nämner vi här som arbetsstation (ws) och den tidpunkt som första spåret i loggfilen finns. Med den informationen kan intrångssignaturerna på nytt hittas bland filerna utan att använda aktivitetsrapporterna. Loggfilerna förvaltas av säkerhetsgruppen på KaU.

5.2.1.1 getadmin

Getadmin är ett program som lägger en användare i administratörsgruppen.

Orsak: På grund av ett problem i en av rutinerna i kärnan kan en global flagga sättas som gör att anrop till API-funktionen NtOpenProcessToken lyckas oavsett rättigheter. De rättigheter som normalt krävs för att använda denna funktion är system- eller administratörsrättigheter. Genom att använda den kan man ta kontroll över processer, däribland WinLogon. Med kontroll över WinLogon kan en ny tråd startas som får dess rättigheter och gör API-anrop som lägger till en användare i administratörsgruppen.

Utförande: Kör programmet och starta om datorn.

Patch: Om service pack 4 installeras tas problemet bort.

Verifierade spår i loggar: ws 1 : 2000-04-18 kl. 12.40.41

5.2.1.2 sechole

Sechole är ett program som lägger en användare i administratörsgruppen.

Orsak: Lokaliserar minnesadressen till API-funktionen OpenProcess, på den lokala maskinen, och modifierar instruktioner där som utför kontroll av rättigheter. Anrop till funktionen lyckas därmed oavsett rättigheter och medför att det är möjligt att ta kontroll över andra processer. Med kontroll över WinLogon kan en ny tråd skapas som får dess rättigheter och gör API-anrop som lägger till en användare i administratörsgruppen.

Utförande: Kör programmet och starta om datorn.

Patch: Om service pack 6 är installerad tas problemet bort.

Verifierade spår i loggar:	ws 5	: 2000-04-26 kl. 1.08.40
	ws 12	: 2000-04-26 kl. 1.21.00
	ws 1	: 2000-05-08 kl. 10.52.54

5.2.1.3 Hämta SAM-fil med reboot

Hämtar den skyddade lösenordsfilen genom att starta om datorn med en startdiskett till MS-DOS och därifrån göra en kopia på den.

Orsak: Det finns möjlighet att starta om datorn från diskett, vilket innebär att det går att köra operativsystem som inte skyddar filer, DOS-miljön har inget skydd av filer.

Utförande: Starta om datorn med till exempel en Windows 95 startdiskett och kopiera SAM-filen. Om den ligger i filsystemet NTFS, kan programmet NTFS-reader användas.

Patch: Enda sättet att förhindra denna attack är att ta bort alla möjligheter att starta om datorn från diskett, cd eller liknande.

Verifierade spår i loggar: Enda spåret detta ger är här att en användare har loggat ut för att lite senare logga in igen, vilket medför att vi bedömer verifiering som allt för chansartat.

5.2.1.4 Knäcka SAM-filens lösenord genom ”bruteforce”

En mängd program finns som kan testa alla kombinationer av lösenord mot SAM-filen. Det finns också en stor mängd ordlistor som kan matchas mot användarnas intresseområden.

Orsak: En tidigare attack har gjort SAM-filen tillgänglig, till exempel genom reboot, se kap 5.2.1.3.

Utförande: Använd förslagsvis l0phtcrack för att utföra bruteforce mot SAM-filen.

Patch: Om SAM-filen hamnar i orätta händer är det enda skyddet att lösenorden är tillräckligt bra konstruerade, att det tar för lång tid att knäcka dem.

5.2.1.5 Sniff av lösenord

Genom att läsa trafiken som passerar på ett nätverk och köra program som känner igen paket relaterade till en inloggning, kan användarnamn och hashade lösenord hittas. Sedan körs bruteforce på det hashade lösenordet.

Orsak: NT skickar användarnamn och hashade lösenord utan ytterligare kryptering vid ”remoteinloggningar”, vidare har en användare fått möjlighet att konfigurera ”promiscuous mode”. ”Promiscuous mode” gör att nätverkskortet läser alla paket som passerar och användaren får möjlighet att sniffa nätet, normalt endast tillåtet som administrator.

Patch: Använda ett starkare autenticeringsprotokoll som exempelvis Kerberos [10].

5.3 Att läsa från händelseloggen

Alla NT-system kommer med en loggningsmekanism och en läsare (Event Viewer) för att kunna presentera loggarna för Administratören, se Bilaga F.

Denna fungerar bra så länge man endast vill använda loggarna för att manuellt se vad som har hänt i systemet.

Problemet kommer när man vill använda sig av dessa loggar i ett försök att detektera misstänkta aktiviteter, som kan tyda på ett intrångsförsök.

Då vi hade för avsikt att använda oss av dessa loggar för att försöka matcha dem mot kända attacksignaturer i databasen (se Kapitel 6 och 7) krävdes dock en effektivare metod för att läsa dem.

5.3.1 Kort bakåtblick

NT erbjuder möjligheten att bland annat öppna och läsa systemloggarna genom vissa fördefinierade API-funktioner.

Vår idé var att skapa en rad funktioner som med ett enkelt gränssnitt mot användaren tillät åtkomst och lätt läsning av dessa loggar genom att anropa API-funktionerna i bakgrunden.

Dessa funktioner skulle sedan ha en central roll i vårt försök att skapa ett Host-Based Intrångsdetekteringssystem, se Kapitel 7.

Andra delar vi ansåg nödvändiga var databasen med attacksignaturerna (se Kapitel 6) och ett huvudprogram som skulle utföra matchningen mellan databasens signaturer och de loggar genererats av systemet.

Några av de API-funktioner som användes var:

- **OpenEventLog / CloseEventLog**

- **ReadEventLog**
- **LookupAccountSid**
- **FileTimeToLocalFileTime**

Tyvärr är inte den struktur som returneras av ReadEventLog speciellt användarvänlig i dess grundutförande.

Det krävdes en hel del jobb innan informationen kunde användas då man tvingades matcha och översätta många av posterna, se Bilaga F.

5.3.2 Tankar under konstruktionen av läsfunktionen

Det första som måste göras är att skapa en pekare till händelseloggen, detta sker med hjälp av funktionen OpenEventLog.

OpenEventLog vill ha UNC-adressen (se bilaga F) till den önskade loggen samt namnet på den logg som ska öppnas (oftast någon av Application, System eller Security).

Funktionen returnerar sedan en filpekare i form av en HANDLE till den valda loggen.

Det första problemet man stöter på är sättet funktionen ReadEventLog egentligen läser loggen. Den logiska gissningen var från början att den skulle läsa en logg i taget tills det inte fanns något mer att läsa, så var dessvärre inte fallet.

ReadEventLog tar bland annat som inargument en char array som ska fungera som buffer för loggarna. Problemet är att funktionen inte lagrar en logg i taget i denna array.

Istället gör den så att den fyller upp arrayen med så många hela loggar den får plats med och returnerar antalet bytes som lästes in.

Det gör inte saken bättre att i princip alla loggar har olika storlek.

Loggarna som ligger lagrade i arrayen är av typen EVENTLOGRECORD och finns beskriven i Bilaga F.

Nu börjar egentligen det stora arbetet med loggen, att bearbeta den så man kan använda innehållet.

Då vi ville ha en lättläst och lättanvänd struktur att läsa ifrån fick vi försöka översätta den lästa ”structen” till en egendefinierad sådan.

Man fick även i detta skede börja fundera lite över vilka delar av strukten som innehåller intressant information ur en säkerhetssynpunkt.

Vi valde preliminärt att den ”struct” som skulle fås vid anropet skulle se ut på följande sätt

```
typedef struct
```

```
{
        char *SIDName;
        char *SIDDomain;
        char **Strings;
        char *SourceName;
        char *ComputerName;
        SYSTEMTIME TimeGenerated;

        DWORD  RecordNumber;
        DWORD  EventID;
        WORD   EventType;
        WORD   NumStrings;
        WORD   EventCategory;
}Record;
```

5.3.3 Motivering av strukturvalet

Vid en enkel jämförelse kan ses att vi tagit bort en hel del poster från originalet, andra poster UserSidOffset, UserSidLength och UserSid behövdes inte längre eftersom de redan använts till att producera de två strängarna SIDName och SIDDomain.

Ett annat huvudbry vi hade under en lång tid var hur vi skulle översätta punkter som t.ex. EventID till den textsträng som den motsvarar. Under arbetets gång kom vi dock fram till att detta bara är onödigt.

Att försöka matcha EventID #528 till strängen ”Successful Logon” fyller inget direkt syfte förutom förtydligande, detta är dock inte speciellt viktigt när man ska försöka matcha mot attacksignaturer. Dessutom kommer en matchning kunna ske mycket snabbare om man jämför siffror istället för siffror och strängar. Vissa strängar är dock nödvändiga, såsom SIDName och SourceName.

Vi tror dock att den Event Viewer som används för att presentera loggarna i NT använder sig av en databas som helt enkelt matchar EventID + SourceName till en sträng. Ett bra exempel på denna databas finns att se på http://www.EventID.Net.

Vi anser att ovanstående fält är de som kan användas vid detektering och de utgör tillsammans de mest väsentliga punkterna i händelseloggen. Med dem som grund är det vår tro att man kan bygga upp ett fungerande IDS.

5.4 Problem

Tidpunkterna som anger start och slut i aktivitetsrapporterna för de enskilda aktiviteterna är i vissa fall inte tillräckligt exakta för att vi tidseffektivt ska hitta spåren i loggfilerna, som ofta blir mycket stora. Däremot gav aktivitetsrapporterna tips om vilka intrång som fungerade respektive inte fungerade.

Vidare hittades spår på att en del av grupperna gjort saker som inte alls redovisats.

6 Intrångsdatabas

Under arbetet har vi samlat på oss en stor mängd data om intrång. Bland annat hur, varför och mot vilka system de fungerar. Dessutom har vi tagit fram signaturer av intrång. All denna information har vi tänkt strukturera i en databas som gör det möjligt att söka efter intrång för att få fram de intrångssignaturer som är ett hot mot ett visst system. Det verktyg vi valt att använda oss av är Microsoft Access eftersom vi tidigare har erfarenhet av detta program. Dessutom kostade Microsoft SQL för mycket pengar, medan det inte fanns dokumentation över Microsoft FoxPro tillgänglig på universitet.

6.1 Syfte och mål

En intrångsdatabas kan användas som en del av ett missbruksdetekteringssystem, men är också användbar som en ren kunskapsbank. Med en bra struktur som har möjlighet att lagra alla sorters signaturer tror vi den skulle kunna bli ett verktyg med potential och värd att utvecklas vidare.

Vårt mål med databasen är att den ska kunna ge svar på olika frågor som berör säkerheten hos ett visst system.

6.2 Design av databasen

Mängden data som vi hittat om intrång har varit stor. Det har varit detaljerade beskrivningar om bristerna i säkerheten som gör intrånget möjligt och hur intrånget i sig fungerar. Vem som officiellt upptäckte intrånget och tidpunkt för detta. Sårbara system, vilka patchar, hotfixes och tillfälliga lösningar som finns och så vidare.

De grundläggande frågor som vår intrångsdatabas ska kunna ge svar på och som vi utgår från när vi gör designen är följande.

- Information om ett visst intrång.
- Signaturen av ett visst intrång.
- Vilka hot som finns mot ett visst system med angivna patchar.
- Vilka hot som finns av en viss klass.
- Vilka patchar som förhindrar ett visst intrång.

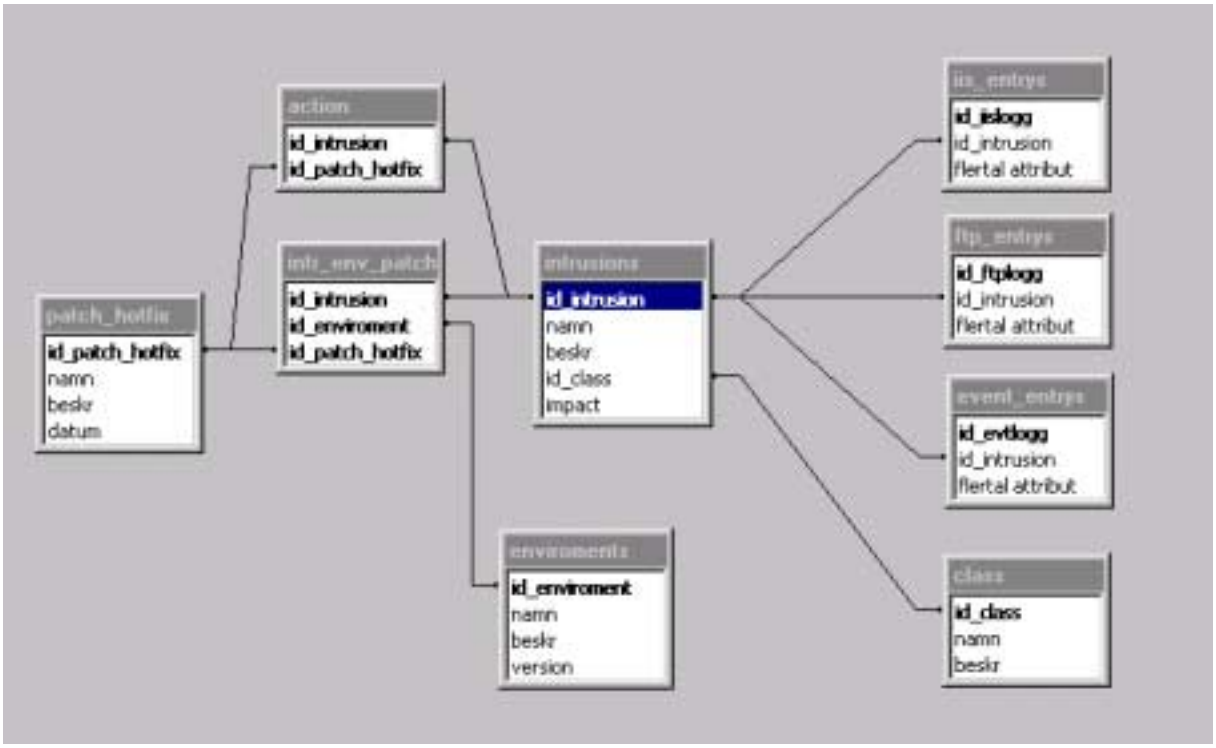
Entiteter är intrång, miljö, klass, patch/hotfix och intrångssignaturer, i vår lösning är antalet tabeller med signaturer dynamisk, då vi vill kunna lägga till en tabell för varje typ av logg. Det finns två relationstabeller, motåtgärd och målmiljö. Motåtgärd skapar ett många-till-många-förhållande mellan intrång och patch/hotfix. Målmiljö tabellen binder intrång, miljöer och patch/hotfix och beskriver därmed vilka system som är sårbara mot ett intrång. I detta fall är ett system en kombination av miljö och patch eller hotfix.

Kort beskrivning av entiteterna.

- Intrång – Alla former av missbruk mot datasystem skall kunna lagras under denna rubrik och innehåller namn, beskrivning av och resultat av missbruk.
- Miljö – Innehåller namn, beskrivning och version på datasystem.
- Klass – Innehåller klassdefinitioner av missbruk
- Patch/Hotfix – Innehåller namn, beskriving och utgivningsdatum på ”patchar” eller ”hotfix”.
- Intrångssignaturer – Innehåller de poster som krävs för att ge en signatur av ett intrång. För varje typ av logg skall det finnas en tabell.

6.3 Implementation av databasen

Nedan kommer vi att ge en mer detaljerad beskrivning av databasens delar. I figur 1 visas grafiskt hur tabellerna är relaterade till varandra.



Figur 6-1: ER-diagram, genererat av Microsoft Access, över intrångsdatabasen.

6.3.1 Entiter

Tabell 6-1: Intrång

Poster	**Attribut**	**Beskrivning**
Id_intrång	Integer	Unik nyckel
Namn	50 chars	Allmän benämning på intrånget
Beskrivning	100 chars	Hur fungerar intrånget
Id_class	Integer	Relation till en klass
Impact	100 chars	Vad som uppnås med intrånget

Tabell 6-2: Miljö

Poster	**Attribut**	**Beskrivning**
Id_miljö	Integer	Unik nyckel
Namn	50 chars	Miljöns namn
Beskrivning	100 chars	Kort beskrivning av miljön
Version	50 chars	Versionen på miljön

Tabell 6-3: Patch/Hotfix

Poster	**Attribut**	**Beskrivning**
Id_patch_hotfix	Integer	Unik nyckel
Namn	50 chars	Namn på patch elller hotfix
Beskriving	100 chars	En beskriving av patchen/hotfixen
Datum	dd/mm yy	Utgivningsdatum

Tabell 6-4: Klass

Poster	**Attribut**	**Beskrivning**
Id_klass	Integer	Unik nyckel
Namn	50 chars	Klassdefinitionens namn
Beskrivning	100 chars	En beskrivning av klassen

6.3.2 Relationstabeller

Tabell 6-5: Motåtgärd

Poster	**Attribut**	**Beskrivning**
Id_intrång	Integer	Sammansatt nyckel
Id_patch_hotfix	Integer	Sammansatt nyckel

Tabell 6-6: Målmiljö

Poster	**Attribut**	**Beskrivning**
Id_intrång	Integer	Sammansatt nyckel
Id_miljö	Integer	Sammansatt nyckel
Id_patch_hotfix	Integer	Sammansatt nyckel

6.4 Problem

Nedan behandlas de problem vi stötte på under arbetet med intrångsdatabasen.

6.4.1 Tveeggat svärd

Verktyg av den här typen kallas ibland för tveeggade svärd eftersom de kan användas på två olika sätt. Systemadministratörer kan använda det för att säkra sina system mot hot som de hittar i databasen, medan en angipare kan använda den för att hitta intrångsmöjligheter. Det finns argument både för och mot en intrångsdatabas, men en majoritet av de verksamma inom området tycker att fördelarna överväger nackdelarna [11]. De är eniga om att kännedom om säkerhetsluckor skall spridas så snabbt som möjligt så att systemansvariga kan säkra sina system.

6.4.2 Loggstruktur

- De loggar som skapas i NT har en dynamisk uppbyggnad som är svåra att återskapa i en databastabell.
- Att hitta en gemensam struktur för en intrångssignatur vore mycket användbart för databasen. Vi har i vår implementation valt att använda en tabell för varje typ av logg. Detta innebär att om en ny typ av logg ska tillföras IDSet krävs att databasen byggs ut med en tabell för just den loggen.

7 Intrångsdetektering

Ett intrångsdetekteringssystem, från engelskans Intrusion Detection System (IDS), kan vara en produkt, ett verktyg eller en kombination av dessa med syfte att upptäcka interna och externa attacker mot ett system samt aktivitet som kan tyda på förberedelser till ett anfall.

Intrångsdetektering är konsten att upptäcka olämplig, felaktig eller icke normal aktivitet.

Det finns många olika typer och de fungerar alla på olika sätt. Gemensamt för de flesta är dock att de letar efter angripssignaturer som är specifika mönster som ofta tyder på felaktig eller misstänksamma avsikter.

7.1 Motiv till att skaffa ett IDS

Utan undantag kommer förr eller senare alla försvarslinjer att penetreras, försvagas eller sluta fungera. Det som du ansåg vara det säkraste systemet någonsin kommer att visa sig ha svagheter som kan utnyttjas och användas till att skada dig, ditt system, ditt företag eller ditt rykte. Yttre försvarspunkter som brandväggar och proxyservers är långt från ofelbara, inre försvar som autenticiering kan gås förbi eller knäckas. Även om detta skulle vara punkter som

inte oroar dig finns också tidsluckan mellan det att en bugg upptäckts och det att den täpps till. Sammanfattningsvis kan man konstatera att säkerhet måste implementeras på djupet.

Ett intrångsdetekteringssystem kan liknas vid en vakthund i ett hus som varnar dig om någon försöker bryta sig in och därefter skrämmer bort eller fångar inkräktaren.

7.2 Common Intrusion Detection Framework (CIDF)

Detta är ett försök att utveckla protokoll och APIs så att människor inom intrångsdetekteringsforskning lättare kan dela erfarenheter och information samt att komponenter ska kunna återanvändas i andra system [12].

Det hela startades av Teresa Lunt när hon arbetade hos Defence Advanced Research Projects Agency (DARPA).

Man har även börjat utvecklingen av ”Common Intrusion Specification Language” (CISL) som man planerar att använda vid intrångsdetekteringssystem och komponenter för att uttrycka information om händelser, attacker och svar på desamma. Nedan följer ett exempel på språket där man beskriver att ’joe’ har försökt eller lyckats att ta bort filen ’/etc/passwd’.

```
(Delete
   (When
      (Time '12:24 15 Mar 1999 UTC')
   )
   (Initiator
      (UserName 'joe')
      (UserID 1234)
      (HostName 'foo.example.com')
   )
   (FileSource
      (FullPathName '/etc/passwd')
      (HostName 'foo.example.com')
   )
```

7.3 Två huvudprinciper för att detektera ett intrång

De två beskrivningarna nedan är från [4].

7.3.1 Avvikelsedetektering

Detta går till så att systemet får lära sig hur en användare beter sig och försöker kartlägga detta beteende i en databas. Varje användare och i flera fall även system får sig varsin profil i databasen. Om en avvikelse från denna profil sker uppdateras profilen alternativt så går larmet beroende på hur stor avvikelsen var. De stora fördelarna med detta system är dels att det uppdaterar sig själv, dels att den kan upptäcka tidigare okända attacker. Nackdelarna är tyvärr flera än fördelarna, åtminstone till antalet. Många falska larm kommer att genereras, och det är ofta svårt att veta vad man detekterat. För varje nytt system eller användare som ska få sig en profil krävs en inlärningsperiod där profilen kan formas korrekt. Det är vidare möjligt då det är ett självlärande system att en angripare hela tiden göra små avvikelser för att till slut få systemet att acceptera stora avvikelser.

7.3.2 Missbruksdetektering

Här förutsätts att man har någon form av databas med händelser som inte tillåts på systemet. Sedan undersöks om någon bryter mot dessa på förhand bestämda reglerna, sker detta så går larmet. Detta brukar kallas för ”default permit”.

Det finns också en annan liknande metod som går under namnet ”default deny”. Den fungerar istället så att en databas innehåller de saker man är tillåten att göra och larmet går om man gör något som inte kan matchas.

”Default deny” är bra eftersom den klarar av även okända intrång och attacker, men har nackdelen att det är svårt att specificera och lagra ett godkänt beteende.

”Default permit” har en väldigt effektiv detektering med låg del falska larm, men den fungerar bara om en given attack finns i databasen vilket innebär att den måste uppdateras mycket ofta.

7.4 Olika arkitekturer

Eftersom intrångsdetektering idag är ett så pass stort område inom datasäkerheten, så finns det självklart många olika lösningar och förslag på det bästa sättet att gå tillväga. De två vanligaste typerna idag är ”Host Based” och ”Network Based”. Det existerar även flera hybrider på marknaden, men de kommer inte att tas upp här.

7.4.1 Host Based

I början på 1980-talet var det fortfarande praxis att undersöka audit-loggarna efter misstänkt aktivitet. Det kom därifrån som ett naturligt steg att försöka skapa ett system som kunde automatisera detta. ”Host Based Intrusion Detection” såg för första gången dagens ljus.

Det var från början inte mycket till system, men det var början på någonting stort inom datasäkerheten.

Dagens versioner är mycket mer automatiserade och sofistikerade, de klarar av att matcha loggar i realtid allt eftersom de skapas mot databaser med kända intrångssignaturer och har till och med möjlighet att slå tillbaka mot en angripare.

”Host Based” IDS använder sig av operativsystemets egna loggar och i vissa fall även av egenproducerade sådana. Så fort någonting läggs till i någon av dessa loggar försöker IDSen matcha detta med alla kända attacker och förbereder när det är lämpligt och möjligt även motåtgärder. Dessa kan vara allt från någonting enkelt, som att meddela administratören, till mer avancerade system, som börjar logga all aktivitet för att samla mer bevis och information för att sedan helt enkelt sparka ut angriparen efter att denne har spårats och nog bevis samlats in.

7.4.2 Stack Based

Detta är det nyaste tillskottet på IDS-marknaden och räknas ofta som ett komplement till ”Host Based” IDS.

Det saknar tyvärr för tillfället en gemensam standard så dess funktionalitet och arbetssätt kan variera mycket mellan olika tillverkare. Det erbjuder dock flera av Network Based IDS tjänsterna eller åtminstone en fin variant av dessa.

”Stack Based Intrusion Detection” arbetar mycket nära TCP/IP-stacken och kan därför undersöka paketen medan de färdas uppåt i stacken.

Genom att undersöka paketen på detta vis kan IDSen dra bort olämpliga paket från stacken innan de kan bearbetas av applikationer eller operativsystem.

För att ett ”Stack Based” IDS ska fungera bra är det lämpligt att den undersöker både inkommande och utgående paket. Då detta är ganska många paket vill man få fram en standard med så lite overhead som möjligt så att systemet ska kunna fungera bra i realtid utan att dra för mycket processorkraft.

7.4.3 Network Based

”Network Based Intrusion Detection Systems” använder sig av nätverkspaketen som datakälla. Detta sker oftast genom att ha ett nätverkskort som kör i s.k. ”promiscous mode” och lyssnar samt analyserar all trafik i realtid.

Först appliceras ett filter som avgör vilken trafik som ska kastas och vilken som ska skickas vidare för eventuell identifiering av attacksignaturer.

Attack-identifieringsmodulen använder sig vanligtvis av mönster-, frekvens- eller avvikelsebaserad detektering av attacksignaturerna.

7.4.4 Fördelar med Host Based Intrusion Detection System

- Större säkerhet i verifikationen av ett lyckat eller misslyckat försök än Network Based IDS då man loggar allt som händer.
- Kan kontrollera specifika systemhändelser som Network Based IDS ofta missar.
- Kan detektera anfall som Network Based IDS missar helt om det inte passerar en uppsatt flaskhals i nätet, eller kanske inte nätet alls.
- Har inga problem med switchade eller krypterade miljöer.
- Nära realtidsdetektering och motåtgärder.
- Kräver ingen extra hårdvara eller dedikerat system.
- Mycket billigare än Network Based IDS.

7.4.5 Fördelar med Network Based Intrusion Detection System

- Man kan skapa flaskhalsar i nätverket där all trafik måste gå igenom. Detta förenklar detekteringen samt minskar kostnaderna eftersom man bara måste kolla ett ställe.
- Detekterar angripare som ”Host Based” IDS missar genom att man tittar på ”packet headers” och därigenom kan få varning om t.ex. DoS-attacker på ett tidigt stadium.
- Försvårar för en angripare att sopa undan spåren eftersom informationen ligger på flera system, så angriparen måste ta kontroll över dels målsystemet dels systemet som har hand om loggarna.
- Kan detektera misslyckade försök och misstänkt beteende på nätbasis.
- Oberoende av operativsystem.

8 Erfarenheter och rekommendationer

Under arbetet har vi kommit fram till följande.

8.1 Loggverktyg

Vi ser ett verktyg som generar egna loggar över systemet som nödvändigt, på grund av att NT's loggsystem är otillräckligt för att upptäcka flera av de angrepp som vi utförde under arbetet. Detta verktyg ska vara fristående och gärna vara en hybrid mellan ”stack-based” och ”host-based”. Det är vidare viktigt att loggarna har en enkel och gemensam struktur. Vi är av åsikten att ”Common Intrusion Specification Language” (CISL) med fördel kan användas som mall. Detta språk är under utveckling i projektet ”Common Intrusion Detection Frameworks” (CIDF).

8.2 Windows NT

Vi tycker inte att Windows NT kan kallas ett säkert system. Det påstås vara skapat med säkerheten i fokus, men vi anser att det i sådana fall har misslyckats. Grundkonfigurationen är fylld med säkerhetsläckor som utnyttjas och exploateras dagligen. Vi anser att det inte finns något enkelt sätt att ordentligt säkra upp ett NT-system, utan det är något som kräver stora kunskaper och erfarenhet inom området. Vidare har vi sett att många av de tjänster som Microsofts programvaror erbjuder är känsliga för ”buffer overflow”-angrepp.

9 Fortsatt arbete

Följande anser vi kan utvecklas vidare.

9.1 Databasen

Databasen kan lagra intrång på ett bra sätt, men än så länge utan tillhörande loggsignaturer. Vi tycker det vore lämpligt att ta fram en gemensam struktur för lagrandet av loggningssignaturer.

9.2 Vår ”EventLogReader”

Vi har tagit fram ett program som kan läsa från händelseloggarna med hjälp av Windows-API funktioner, se Kapitel 5.3. Detta program behöver testas ytterligare för att säkra dess stabilitet. Den är i dagsläget utformad för att ta fram den information vi nu ser som viktigast för att kunna detektera ett angrepp. Detta kan behöva modifieras för att kunna användas tillsammans med databasen, vilket inte kan avgöras förrän strukturen för loggningssignaturerna för densamme är fastställd.

9.3 Grafiskt gränsnitt mellan vår IDS och databasen

Vi har kontrollerat möjligheten att använda Microsoft Visual C++ för att skapa ett grafiskt användargränssnitt. Detta gränssnitt kommer ha som primär uppgift att ta hand om kommunikationen mellan användare och IDS. Gränssnittet kommer att hämta data från användaren och sedan ta ansvar för att IDS-programmet utför den aktuella förfrågan. Gränssnittet fungerar således som länken mellan användare och IDS. Vi har utfört en del tester för att verifiera att kommunikationen mellan en kommande IDS-applikation och databasen fungerar på ett tillfredsställande sätt.

10 Resultat

Vi har lyckats med att skapa en struktur för information om intrång och designat en databas efter den. Vi har kontrollerat arbetet i de tidigare utförda projekten. Vi har undersökt möjligheterna med att använda NTs inbyggda loggningsmekanism för intrångsdetektering och skrivit ett program som läser loggar. Vi har beskrivit de vanligaste sätten att detektera angrepp och vilka metoder som används vid angrepp. Tyvärr har vi inte kunnat fastställa en bra struktur på en intrångssignatur, dels på grund av tidsbrist men också för att vi inte hade tillräckligt med material för att veta vilken information som är nödvändig i en sådan. Vi arbetade med målsättningen att utveckla ett intrångsdetekteringsverktyg. Därför har vi även tagit fram ett användargränssnitt till verktyget och kommunikation till intrångsdatabasen med hjälp av Microsoft Visual C++. Med dessa delar var det sista som behövdes för att slutföra arbetet med en struktur för intrångssignaturer för att kunna lagra dem i databasen. Vi hade tagit fram de loggar som genererades av ett flertal angrepp och det var utifrån dem vi skulle ta

fram strukturen, men informationen från angreppen var alltså inte nog för att kunna fastställa en lämplig struktur.

Referenser

[1] Stefan Lindskog och Erland Jonsson, Different Aspect of Security Problems in Network Operating Systems, ISSEA, Orlando, Florida, USA, 2002.

[2] Stuart McClure, Joel Scambray och George Kurtz. Hacking Exposed, third edition. Osborne McGraw-Hill, 2001.

[3] http://www.isi.edu/gost/cidf - Hemsida för Common Intrusion Detection Framework.

[4] Föreläsningsanteckningar av Hans Hedbom för kursen Tillämpad datasäkerhet (DAVC17), 2002.

[5] W. Richard Stevens, UNIX Networking Programming, second edition. Prentice Hall PTR, 1998.

[6] Nate Miller, Microsoft IIS Unicode Exploit, Lucent Technologies Worldwide Services, 2001.

[7] http://online.securityfocus.com/library Smashing the Stack for Fun and Profit, by Aleph One, 1996.

[8] Lee Hadfield, Dave Hatter och Dave Bixler.Windows NT Server 4 Security handbook. QUE, 1997.

[9] Ira S. Winkler och Brian Dealy. A Case Study in Social Engineering. 5th Usenix, 1995.

[10] http://web.mit.edu/kerberos/www/ - Kerberos: the Networking Authentication Protocol.

[11] Stefan Lindskog. Observations on Operating System Security Vulnerabilities. Chalmers University of Technology, 2000.

[12] http://www.isi.edu/gost/cidf/ Common Intrusion Detection Framework, 1998.

[13] http://www.cultdeadcow.com/cDc_files/cDc-351/.

[14] James D. Murray, Windows NT Event Logging. O'Reilly, 1998.

[15] http://msdn.microsoft.com.

[16] Charles B. Rutstein. Guide to Windows NT Security. McGraw-Hill 1997.

[17] http://www.nic-se.se/ - Network Information Centre Sweden AB.

A Laborationsspecifikationer

Tillämpad datasäkerhet (DAVC17) — Projekt

Introduktion

Projektet går ut på att praktiskt undersöka säkerheten i ett Windows NT-system. Detta kan göras på flera olika sätt, och vår metod består av att försöka bryta sig in! **Arbetet får tidigast påbörjas måndagen den 17 april och ska vara slutfört senast söndagen den 14 maj**. Under dessa fyra veckor ska rapportblanketter om hur arbetet fortskrider kontinuerligt lämnas in.

Bakgrund

När man ska försöka beskriva hur säkert ett datorsystem är finns det egentligen inga riktigt bra metoder tillgängliga. Ett förslag har framförts inom EU-projektet *PDCS (Predictably Dependable Computing Systems*) där man i stället för att försöka beskriva säkerhet som en funktion av tiden i stället försöker beskriva säkerhet som en funktion av nerlagt arbete (eng. *effort*). Man antar helt enkelt att ju mer arbete man lägger ner, desto större är chansen att bryta sig in i ett system. Meningen med projektarbetet är därför dels att ge praktisk erfarenhet om hur Windows NT skyddssystem fungerar genom att helt enkelt leta efter problem i systemet, och dels att verifiera huruvida *effort* är en lämplig *approach* till problemet.

Vitsen med att leta igenom ett Windows NT-system är att man tvingas att förstå dess skyddssystem. Det kommer med största sannolikhet att bli flera laborationsgrupper som inte alls lyckas hitta någon väg in i systemet medan andra hittar flera vägar. En del grupper känner säkert till olika intrångssätt redan innan vi startar, men deras arbete börjar egentligen på fullt allvar först när dessa kända sätt är avklarade. Det är heller inte säkert att erfarenhet ger bäst resultat i alla lägen. Med lite klurighet och fantasi kommer man minst lika långt! Alla säkerhetsintrång är ju fortfarande intrång oavsett om de var enkla eller mer komplicerade att genomföra.

Beskrivning av systemet

Laborationssystemet består av 12 stycken arbetsstationer som heter **`elab-ws1`** till **`elab-ws12`**. Dessa är i sin tur kopplade till servrarna **`tdspdc`** och **`tdsbdc`**. På samtliga maskiner körs Windows NT version 4.0 med service pack 3 (SP3).

Med bakgrund av att de flesta säkerhetsrelaterade problemen i industrin orsakas av så kallade *insiders*, så är tanken att även detta jobbet ska utföras på liknande sätt. Därför har ni tilldelats konton på precis samma sätt som görs i andra kurser och ni använder institutionens standardsystem utan att några säkerhetshöjande åtgärder har vidtagits!

Rapportering

Innan projektarbetet påbörjas skall varje student fylla i och lämna in ett exemplar av blanketten "Bakgrundsrapport". Avsikten med denna blankett är att ge ett underlag för bedömning av varje students förkunskaper och tidigare erfarenheter. Detta möjliggör en "viktning" av parametern *effort* vid den senare utvärderingen.

Under själva projektarbetet skall alla aktiviteter rapporteras på blanketten "Aktivitetsrapport". Detta dels för att man i efterhand ska kunna beräkna just variabeln *effort* och se hur bra den verkar, och dels för att vi ska kunna bedöma varje laborationsgrupps arbetsinsats. Påbörja en aktivitetsrapport omedelbart när ni gör något arbete som kan relateras till projektarbetet. Det kan ibland vara så enkelt som att ni ligger hemma i sängen och funderar ett tag, men rapportera då hur mycket tid som lades ner! Tidigare erfarenheter visar att det är väldigt lätt att underskatta sin egen arbetsinsats! När en aktivitet är avslutad, skicka in rapporten så snart som möjligt till laborationshandledaren så att han har kontroll på hur arbetet fortskrider och på vad som pågår med systemet.

Efter avslutat projektarbete ska varje grupp skriva en slutrapport som sammanfattar ert arbete. Denna ska kort och koncist ange hur ni har disponerat tiden, vilka huvudmetoder ni arbetat efter, vilka eventuella säkerhetsproblem ni har funnit och vad ni kommit fram till angående Windows NT’s säkerhetsfilosofier. Där får gärna också stå allmänna kommentarer om projektarbetet, om vad ni har lärt er, vad ni fått ut (eller inte fått ut) av arbetet och hur det i stort har fungerat. Redogörelsen bör rymmas på ca 10 sidor.

B Rapportmallar

B.1 Bakgrundsrapport

Tillämpad datasäkerhet (DAVC17) — Projekt

BAKGRUNDSRAPPORT

1. Gruppnummer: `tds_____` Gruppmedlem: ❑ A ❑ B
2. Årskurs: _____
3. Ungefär hur många datainriktade kurser har du läst?

   Hur är det med:

   C och UNIX? ❑ Ja ❑ Nej

   Distributed Systems? ❑ Ja ❑ Nej

   Distributed Systems Design? ❑ Ja ❑ Nej

   Kryptografi? ❑ Ja ❑ Nej

4. Har du arbetat professionellt med datorer (mer än sommarjobbat)? ❑ Ja ❑ Nej

   Hur länge?

   Med vad?

   Med Windows NT?

   Med säkerhet?

5. Har datasäkerhetsfrågor intresserat dig tidigare? ❑ Ja ❑ Nej

   Om ja, varför?

   Har du testat säkerheten någonstans tidigare?

   Hur mycket tid har du ägnat åt datasäkerhet tidigare?

6. Använder du datorer på fritiden? ❑ Ja ❑ Nej

   Om ja, ungefär hur många timmar per vecka?

Till vad?

Vad för slags system?

7. Varför har du valt den här kursen?

8. När hörde du talas om det här försöket första gången?
   Hur mycket visste du om projektarbetet innan kursen började?

9. Vilken kunskapsnivå i datasäkerhetsfrågor bedömer du att du har i förhållande till en "medelstudent" i årskurs 3? Ange ett värde 1-10, där 1=bland den 10% sämsta, 2=mellan 10 och 20%, osv.)

10. Hur mycket tid har du lagt ner på att fundera över dataintrång under månaden närmast försökets början?

11. Vilka idéer har du just nu för hur man angriper målsystemet?

12. Övriga synpunkter? Vad mer kan vara värt att veta om dig?

B.2 Aktivitetsrapport

Tillämpad datasäkerhet (DAVC17) — Projekt

Aktivitetsrapport

```
+-------------------------------------------------------------------+
|                       Aktivitetsrapport                           |
+-------------------------------------------------------------------+

Gruppnummer:
Datum:
Rapportnummer:
Arbetsstation:

1.  Namn eller typ av aktivitet:

2.  a) Om det är en fortsättning på en tidigare attack, ange dess
       rapportnummer:

    b) Om det är en ny attack, beskriv den:

3.  När utfördes aktiviteten (ange flera delaktiviteter om nödvändigt)
    Datum [mm-dd]:       Starttid [hh:mm]:       Sluttid [hh:mm]:


Sammanfattning
--------------

4.  Total arbetstid [hh:mm]        A:       B:       A+B:

5.  Uppskattad total on-line-tid:  A:       B:       A+B:

6.  Vilka andra resurser förutom arbetsstationen användes (böcker,
    personlig hjälp, etc.):
    A:
    B:
    A+B:

7.  Varför avslutade ni aktiviteten?


Ifylles om du uppnådde något, t ex ett intrång
----------------------------------------------

8.  a) Vad uppnådde ni?

    b) När uppnådde ni det? Ange tid [hh:mm] eller varaktighet:

    c) Förklara hur ni uppnådde det:

    d) Var någon tidigare aktivitet nödvändig för att uppnå det?
```

```
Avslutande frågor och övriga kommentarer
----------------------------------------

9.  Har ni några framtida angreppsplaner? Vilken/vilka?

10. Andra kommentarer:
```

B.3 Slutrapport

Tillämpad datasäkerhet (DAVC17) — Projekt Slutrapport

Omfattning

Slutrapporten bör rymmas på ca 10 A4-sidor **exklusive** titelsida och innehållsförteckning.

Innehåll

Vi förväntar oss en tydlig och välstrukturerad rapport. Tänk på att skriva rapporten på ett sätt som gör den intressant att läsa för andra än lärarna på kursen. Nedan presenteras ett förslag till rapportstruktur.

<Titelsida med namn, personnummer och gruppnummer>

Innehållsförteckning

1. Sammanfattning (ca 1 sida)

2. Informationssökning (max ½-1 sida)

3. Intrångsförsök

3.1 <Titel på intrångsförsök 1>

3.1.1 Inledning

3.1.2 Tillvägagångssätt

3.1.3 Resultat

3.1.4 Kommentarer

3.2 <Titel på intrångsförsök 2>

3.2.1 Inledning

3.2.2 Tillvägagångssätt

3.2.3 Resultat

3.2.4 Kommentarer

…

4. Slutsatser (½-1 sida)

Referenser

[1] <Författare. Titel. Förlag e dyl, utgivningsår.>

[2] <Författare. Titel. Förlag e dyl, utgivningsår.>

…

Bilaga A – <Titel på bilaga A>

Bilaga B – <Titel på bilaga B>

…

C Projektregler

Tillämpad datasäkerhet (DAVC17) — Projekt

Regler för genomförandet

I princip är allt tillåtet, men av praktiska skäl måste vissa regler och begränsningar finnas:

Allmänna regler

1. Hjälp kan fås av laborationshandledaren, både i tekniska och i praktiska frågor. Han kan t ex låna ut litteratur eller viss utrustning som ni behöver.
2. Av naturliga skäl så får ni bara försöka att bryta er in i laborationssystemet, dvs i servrarna **`tdspdc`** och **`tdsbdc`** samt i arbetsstationerna **`elab-ws1`** – **`elab-ws12`**. Ger ni er på något annat system gör ni er skyldiga till ett lagbrott och kan straffas enligt strafflagen.
3. Vid osäkerhet om vad som är tillåtet eller vad som är lämpligt, ska laborationshandledaren tillfrågas. Laborationshandledaren utgör en helt neutral resurs tillgänglig just för er!

Regler för realism

4. Arbeta oberoende av andra grupper. Samarbete mellan grupperna förstör själva grundidén med arbetet.
5. Skulle ni lyckas få tillgång till en annan grupps konto, så är detta naturligtvis ett lyckat intrång. Men leta inte bland deras filer för att få fler idéer!
6. Sprid inte kännedom om er verksamhet till studenter i andra kurser, framför allt för att de då kan bete sig annorlunda till nackdel för er (som t ex att skydda sina filer extra bra). Tänk också på att personer som inte deltar i kursen och som vill försöka bryta sig in gör detta helt på egen risk och utan tillstånd.

Begränsningar för att systemet ska fungera

7. Det är inte tillåtet att fysiskt ge sig på utrustningen, t ex att förstöra eller modifiera denna.
8. Ett intrång får *inte reducera tillgängligheten* till systemet för de normala användarna, t ex genom att lasta ned maskinerna. Systemet kommer att användas i andra kurser parallellt med att detta projektarbete genomförs och de andra laboranternas arbete *får inte störas*. Det är alltså viktigt att systemet fortsätter att fungerar normalt trots era intrångsförsök.
9. Om ni har någon idé om hur tillgängligheten kan minskas *utan att ni själva blir upptäckta* eller har någon annan bra ide till ett försök, vars konsekvenser ni inte kan överblicka, så kontakta laborationshandledaren innan ni genomför försöket. Han kanske tycker att idén är värd att pröva men att det bör ske under kontrollerade former, t ex på kvällstid e dyl.
10. Att söka aktiv hjälp via datornät, som t ex att ställa frågor via News, är inte tillåtet. Huvudanledningen är givetvis att vi inte vill att obehöriga personer från andra ställen ska börja undersöka systemet och försöka bryta sig in.
11. Att aktivt göra konstigheter mot nätverket kräver laborationshandledarens tillstånd. Detta är nödvändigt för att skydda andra maskiner som sitter på vårt nätverk. Laborationshandledarens kan t ex tillfälligt isolera laborationssystemet från resten av nätet om detta skulle behövas.
12. Av rent praktiska skäl är det inte tillåtet att köra program på flera av labbets arbetsstationer samtidigt. Lämnar ni dessutom en arbetsstation som kör något av era program, så måste maskinen vara användbar för nya laboranter som söker upp den (den ska t ex inte vara för nedlastad för att kunna användas).
13. Om ni lyckas få administratörsrättigheter så ska ni inte utnyttja detta på något sätt. Läs inte filer som är lässkyddade, skapa inte nya säkerhetshål, etc. Detta är inte för att skydda något hemligt utan för att ni ska kunna fortsätta med nya attacker mot systemet och se om ni hittar fler problem. Då ska ni naturligtvis bara ha sådan kännedom om systemet som en vanlig användare har och inte kunskaper som ni fick då ni var *Administrator*.

Regler för rapportering

14. En ny aktivitetsrapport ska påbörjas så snart en ny aktivitet startas. Om ett längre uppehåll görs, om ni övergår till en annan strategi (dvs till en annan aktivitet) eller om ni lyckas bryta er in så ska den gamla rapporten avslutas och en ny påbörjas.
15. Skicka in avslutade aktivitetsrapporten omedelbart, helst via e-mail, till laborationshandledaren. Detta dels för att förhindra att någon rapport ska komma bort men framför allt för att laborationshandledaren ska kunna koordinera aktiviteter och veta vad som händer med systemet för att t ex lättare kunna hitta problemet om något kraschar.
16. Även om andra datorsystem i vissa fall skulle gå att använda för någon aktivitet, så använd labbsystemet till så många aktiviteter som möjligt. Här är *accounting* påslagen så att använd CPU-tid m m loggas. Används ett annat system tappar vi den här möjligheten. Tänk också på att mycket av verksamheten inte är tillåten på andra system!

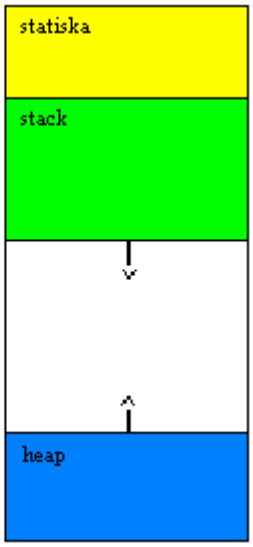
D Buffer overflow

”Buffer overflow” uppstår när en sträng matas med fler tecken än vad som är allokerat för den i minnet och leder till att de minnesadresser som finns efter strängen blir överlagrade med nytt innehåll. På grund av det sätt som minnet hanteras av datorn är det på detta sätt möjligt att skriva över en minnesadress som pekar ut vart i programmet processen skall fortsätta exekvera. Om den skrivs över med en adress som ligger utanför den plats i minnet som programmet har förfogande över får man ett ”access denied”-fel. Därför är det bara möjligt att gå till och utföra instruktioner som finns i programmet, men genom att fylla en sträng med programkod och sedan styra processen till att fortsätta sin exekvering i den strängen går det att ta kontroll över processen och de rättigheter som processen har. Detta är vad angripare gör när de använder ”buffer overflow” för att göra intrång.

D.1 Minne och run-time stack

För att skriva en ”exploit” som utnyttjar ”buffer overflow” krävs kunskap om hur ”run-time-stacken” fungerar och grundläggande C-programmering. Stacken växer antingen uppåt eller nedåt i minnet, beroende på hur den är implementerad, vi har valt att låta bilderna illustrera en som växer nedåt.

Minnet är uppdelat i tre delar, det statiska minnet, stacken och heapen. När ett program exekverar lagras dess variabler i något av dessa tre områden beroende på vilken livslängd variabeln har. Globala och statiska variabler samt programkoden lagras i det statiska minnet, lokala variabler lagras i stacken och heapen används till variabler explicit allokerade med kommandona ”new” och ”malloc”, mellan stack och heap finns det lediga minnesutrymmet i datorn. Det statiska minnet har en fast storlek, medan stack och heap växer mot varandra tills det lediga minnet tar slut, se Figur D-1: Grafisk beskrivning av minnet. Det är hur stacken är uppbyggt som vi är intresserade av. Namnet run-time-stack härleder från att den är implementerad med en stack-funktion och minne som allokeras här gör det under exekveringstid.



Figur D-1: Grafisk beskrivning av minnet

Ett urval av termer som används angående stacken är:

- stack frame (det block av minne som skapas när en funktion anropas)
- SP – Stack Pointer (register som pekar på toppen av stacken)
- FP – Frame Pointer (register som pekar på en bestämd position i en stackram)
- IP - Instruction Pointer (adressen till den instruktion i det statiska minnet som ska exekveras)

Skälet till att använda en stack, är att den gör det möjligt för programspråk att använda funktioner. En funktion gör det möjligt att hoppa till en annan del av koden utan att glömma bort varifrån hoppet gjordes, och det blir därför möjligt att komma tillbaks när funktionen är klar. I äldre programspråk, som till exempel gwbasic, kunde inte funktioner användas. Alternativet var ”goto’s”, vilket leder till mycket stora och svårlästa program och som bryter mot dagens vanligaste programmeringsstilar, funktions- och objektorientering.

När ett funktionsanrop görs sker följande i nämnd ordning.

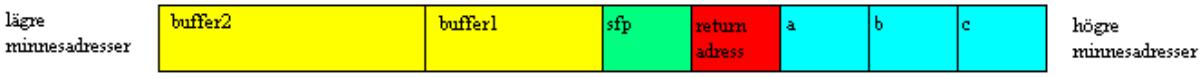
- Parametrarna läggs på stacken.
- Instruction Pointer (IP) läggs på stacken och kallas där return adress eller RET (för att veta varifrån funktionen anropas)
- Frame Pointer (FP) läggs på stacken och kallas där Saved Frame Pointer (SFP).
- Nuvarande Stack Pointer (SP) kopieras till Frame Pointer (FP).

- Stack Pointer (SP) flyttas sedan framåt för att ge plats till lokala variabler.

Som exempel använder vi programmet nedan och Figur D-2: Stackram, visar den ram som genereras i stacken då funktionen function anropas. Exemplet är direkt taget från [7].

```
void function(int a, int b, int c){
        char buffer1[5];
        char buffer2[10];
}

void main() {
        function(1,2,3);
}
```



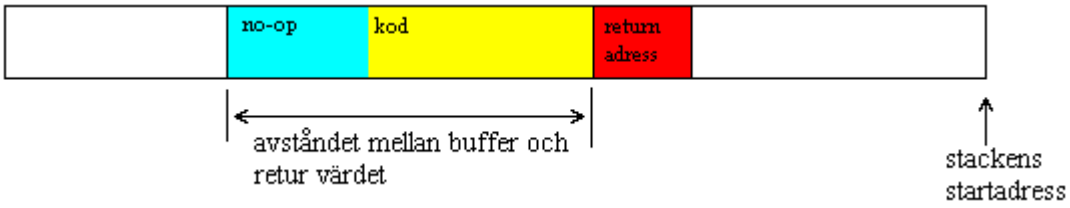
Figur D-2: Stackram

I Figur D-2: Stackram, syns tydligt hur ”return adress”-värdet skulle bli överlagrat om någon av buffer1 eller buffer2 skulle matas med ett för stort antal tecken. Som dataingenjörer har detta skett omedvetet vid ett antal tillfällen under diverse laborationer. Det har då blivit en slumpartad returadress som hamnat utanför det tilldelade minnet, men vår insikt är nu att varje gång ett sådant fel kunde inträffa fanns möjligheten att utföra en ”buffer overflow”-exploit.

D.2 Exploit

På grund av den stora potential som finns i ”buffer overflow”-attacker läggs mycket tid ner på att hitta dem. Genom att skriva in långa teckensträngar när program ber om sådana hoppas man att det ska generera någon form av felmeddelande som ”access denied” eller förbjuden åtgärd. När det inträffar startas programmet i en debugger och strängen matas in på nytt. Om det är en ”buffer overflow” kommer, efter felmeddelandet, ”instruction pointer” registret innehålla en adress som härleder från strängen. Till exempel en sträng med endast a, vars hexadecimalavärde är 61, ger adressen 0x61616161. Efter det vill man veta hur långt

strängens början är ifrån RET adressen. Det tas fram genom att rent praktiskt testa vilken minimal längd strängen har för att skapa ”buffer overflow”. Nu återstår att skriva ett program samt hitta adressen som programmet startar på när det finns på stacken. Eftersom stackens startadress alltid är densamma för alla program kan vi utgå från den och helt enkelt testa oss fram tills den rätta adressen hittas. För att öka chanserna att träffa rätt skapas en större träffyta genom att lägga till ”no operation” (no-op) instruktioner före koden, se Figur D-3:Stackram med träffyta..



Figur D-3:Stackram med träffyta.

När en adress till ”no-op” fältet hittats är det ett fungerande ”buffer overflow exploit”. Det finns dock ett antal problemområden som vi inte tagit upp. Problem som kan förekomma är om null-tecken behövs i koden, men om de skrivs in så termineras istället strängen eller till exempel att bufferten är liten.

D.3 Tillägg

Det finns fler typer av overflows och sättet att utnyttja dem kan variera. Den förklaring som vi tillhandahåller om utförandet är knapphändig, mer detaljerad information finns i [7] och [13].

E Microsoft Internet Information Server 4.0

Internet Information Server (IIS) är bland annat webbserver för Windows från Microsoft. Den har två typer av tjänster, externa och interna. Externa tjänster tillhandahåller någon form av tjänst utåt från servern sett, medan interna verkar inom servern. Tjänster som version fyra tillhandahåller är följande:

Externa:

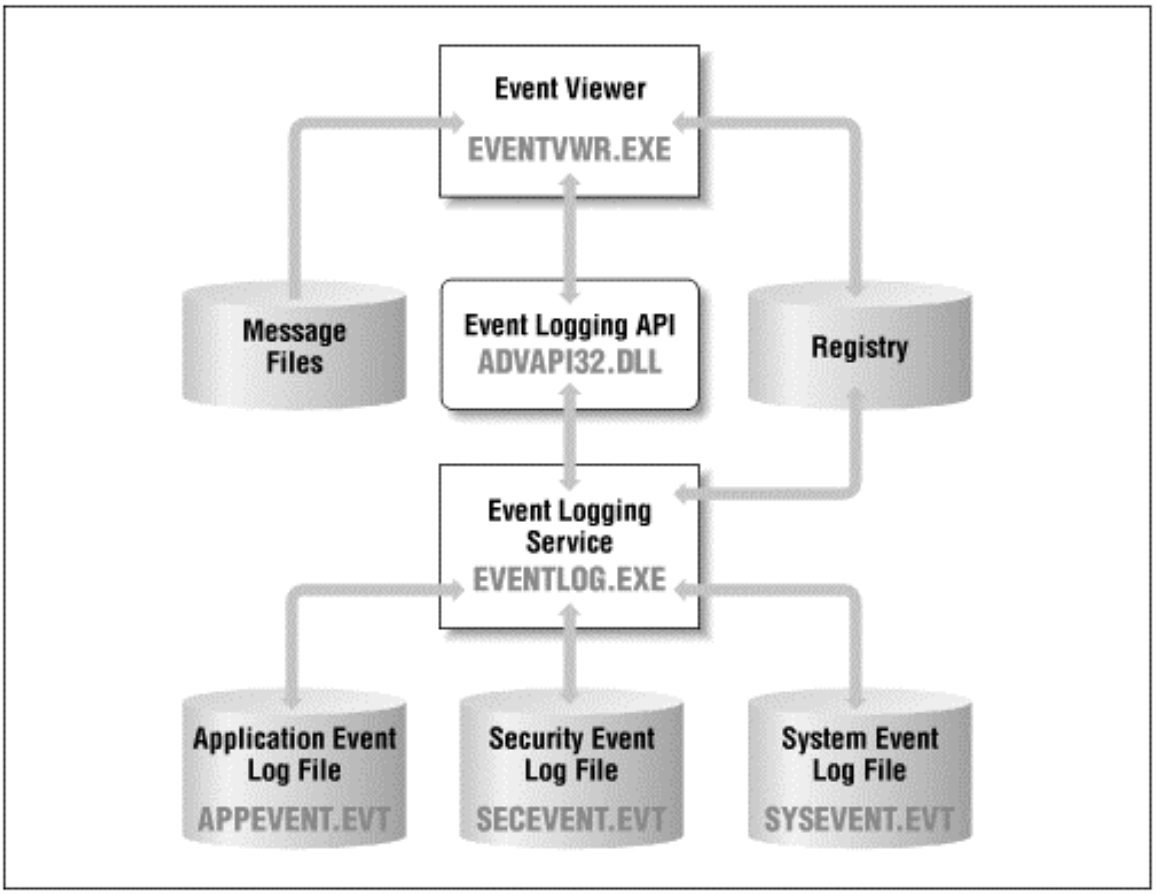
- Web-server (hyper text message protocol) för distribution av webbsidor.
- FTP- server (File Transfer Protocol) för distribution av filer.
- SMTP-server (Simple Mail Transfer Protocol) för distribution av e-post.
- NNTP-server (Network News Transfer Protocol) för att sätta upp nyhetsgrupper.

Interna:

- Transaktionsserver för säker databashantering.
- Indexserver för att tillhandahålla sökmöjligheter.
- Certifikatserver för hantering av nycklar och krypterad kommunikation.
- Siteserver för analysering av webbplatsen.

F Teknisk beskrivning av EventLogRecord strukturen

Denna bilaga är framtagen med information från [15] och [14].



Figur D-4:Sambandet mellan delarna som utgör händelseloggaren

När Windows NT startas upp startas SCM (Service Controll Manager), som i sin tur startar alla de tjänster som är konfigurerade för automatisk uppstartning. En av dessa tjänster är händelseloggaren (EVENTLOG.EXE). När en användarprocess vill komma åt händelseloggarna måste den göra ett anrop till händelseloggningsmekanismen som utför den begärda operationen.

Begäran görs genom ett anrop till en eller flera av de funktioner definierade av händelseloggningens API.

Dessa API-funktioner görs tillgängliga för Win32 processer genom *%SystemRoot%\SYSTEM32\ADVAPI32.DLL* Dynamic Link Library (DLL).

Denna DLL innehåller flera andra APIs, bland annat de som används av SCM, NT security och registret.

Händelseloggningens API är den enda mekanismen som kan hämta och spara händelser till Windows NTs händelseloggare.

När operativsystemet, ett program eller en hårdvarudel rapporterar en händelse skickar den egentligen en rapport till händelseloggningstjänsten. Denna tjänst eller mekanism lagrar sedan all den information som är associerad med händelsen som en struktur, Figur F-5: Eventlogrecord strukturen, i en av de tre händelsefilerna som lagras lokalt på systemdisken (eller en egendefinierad sådan).

```
typedef struct _EVENTLOGRECORD {
  DWORD  Length;
  DWORD  Reserved;
  DWORD  RecordNumber;
  DWORD  TimeGenerated;
  DWORD  TimeWritten;
  DWORD  EventID;
  WORD   EventType;
  WORD   NumStrings;
  WORD   EventCategory;
  WORD   ReservedFlags;
  DWORD  ClosingRecordNumber;
  DWORD  StringOffset;
  DWORD  UserSidLength;
  DWORD  UserSidOffset;
  DWORD  DataLength;
  DWORD  DataOffset;
  //
  // Followed by:
  //
  // TCHAR SourceName[]
  // TCHAR Computername[]
  // SID   UserSid
  // TCHAR Strings[]
  // BYTE  Data[]
  // CHAR  Pad[]
  // DWORD Length;
  //
} EVENTLOGRECORD, *PEVENTLOGRECORD;
```

Figur F-5: Eventlogrecord strukturen

Nedan kommer varje fält i ovanstående struktur att beskrivas och vissa kommentarer till uppbyggnaden kommer att ske.

Length

Den totala storleken i bytes av hela den aktuella loggen. Detta värde lagras även i slutet av loggen, detta för att det ska bli enkelt för ”Windows Event Viewer” att stega mellan loggarna.

Reserved

Ett fält som bara används av Windows loggningsservice. Det innehåller byte värdena 0x4c, 0x66 och 0x65 och används troligen som magiska filidentifikationsnummer.

RecordNumber

Det absoluta indexnumret för den aktuella loggen. Den första loggen som skapas kommer att ha indexnummer 1. Skrivs sedan den loggen över, kommer den första loggen att ha ett indexnummer större än 1.

TimeGenerated

Antalet sekunder från midnatt den 1 januari 1970 till dess att den aktuella loggen genererades.

TimeWritten

Antalet sekunder från midnatt den 1 januari 1970 till dess att den aktuella loggen skrevs ner. Detta värdet kan skilja sig från TimeGenerated fältet av flera orsaker, en av dessa är möjligheten att logga över nätverket.

EventID

Ett 16-bitars värde som oftast ligger inom gränsvärdena 1 till 9999 och fungerar som identifierare för den specifika händelsen. Tyvärr är det inte speciellt enkelt eller självklart att översätta siffran som ligger lagrad till den beskrivande strängen som den representerar.

Nedan följer ett par vanliga exempel på säkerhetshändelser.

Tabell F-1: Exempel på säkerhetshändelser

Kategori	**Händelse**	**Event ID**
System Event	System Restart	512
	System Shutdown	513
Logon/Logoff	Sucessful Logon	528
	Unknown Username or Bad Password	529
	User Logoff	538
	Account Locked Out	539
Object Access	Object Open	560
	Handle Allocated	561
Detailed Tracking	New Process Has Been Created	592
	Process Has Exited	593
Account Management	User Account Created	624
	User Account Type Changed	625a
	User Account Password Set	628a

EventType

Ett värde som indikerar typen av händelse lagrad i loggen. EventType kan ha följande värden

Tabell F-2: De fem olika ”event-typerna”

Name	Value
EVENTLOG_ERROR_TYPE	1
EVENTLOG_WARNING_TYPE	2
EVENTLOG_INFORMATION_TYPE	4
EVENTLOG_AUDIT_SUCCESS	8
EVENTLOG_AUDIT_FAILURE	16

NumStrings

Antalet strängar som finns lagrade mellan UserSid och Data i händelseloggen.

EventCategory

Beskriver den kategori som EventID ovan faller under. Ett vanligt scenario kan vara ett EventID lika med 529, vilken enligt EventID-tabellen ovan innebär att någon gjort ett misslyckat inloggningsförsök. EventCategory skulle i detta fall vara Logon/Logoff.

Tyvärr är det inte lika enkelt att få fram detta direkt eftersom EventCategory sparas som ett DWORD. För att få fram rätt kategori som en läsbar textsträng måste man därför antingen manuellt slå upp EventID eller skapa en så kallad ”Message file”, och med hjälp av den plus ett antal funktioner få fram strängen.

ReservedFlags

Ett bitfält som endast används av Händelseloggaren.

ClosingRecordNumber

Är ytterliggare ett reserverat nummer. Vanligtvis är det ett värde som identifierar den händelse som ”avslutar” den aktuella händelsen. Som exempel kan man ta en händelse ”öppna fil”, då skulle den avslutande händelsen vara ”stänga fil”. Händelser som associeras med öppna och stäng operationer brukar kallas för ”händelsepar”.

Fältet verkar dock inte användas i den nuvarande implementering av Händelseloggaren.

StringOffset

Antalet bytes från strukturens början tills det att första strängen startar.

UserSidLength

Längden av den aktuella strukturens SID (Security Identifier) i bytes. Används tillsammans med UserSidOffset när man vill få fram vilken användare som orsakade en viss händelse. Detta sker genom ett API anrop till LookupAccountSid.

UserSidOffset

Samma användningsområde som StringOffset, men denna anger istället avståndet mellan början av strukturen och början på UserSid.

DataLength

Längden på den binära data som är associerad med strukturbeskrivningen.

DataOffset

Anger i bytes avståndet från början av strukturen till början av datan.

SourceName

Namnet på den som rapporterade händelsen. Strängen sparas som en Unicode (WCHAR) och är NULL-terminerad. De flesta säkerhetsrelaterade händelser brukar ha ”Security” i detta fält. Går man vidare till Applikationsloggen brukar man hitta namnet på programmen som genererat händelsen som loggats.

ComputerName

Namnet på den dator där händelsen inträffade. Även denna sparas som en Unicode-sträng och är NULL-terminerad. Har man inte en central loggningsenhet i ett nätverk brukar den här posten ofta innehålla namnet på den egna datorn.

UserSid

Källidentifieraren av användaren som rapporterade eller orsakade händelsen. Detta värde används ofta för att referera till namnet av en användare i SAM-databasen. Detta fält kan vara tomt om händelsen inte orsakades av en användare.

Strings

Här följer NumStrings-strängar varandra. Alla är NULL-terminerade. Detta fält kommer inte existera om det inte finns några parametersträngar associerade till händelsen.

Data

Den binära datasträngen som är associerad med händelsen. Detta fält kommer inte att existera om det inte finns någon binär data associerad med händelsen.

Pad

Detta fält innehåller ett till tre ASCII NULL bytes för att förlänga strukturen så mycket att dess totala storlek blir jämt delbar med fyra.

Felkoder

Alla händelseloggningsfunktioner returnerar ett icke-NULL värde vid lyckad operation och ett NULL-värde vid en misslyckad operation. För de funktioner som returnerar ett HANDLE är detta en indikation på om en händelselogg kunde öppnas eller inte.

För resterande funktioner fungerar TRUE som en indikation på att den lyckats och FALSE på att den misslyckats.

Mer information om ett fel kan ofta fås genom att kalla på GetLastError omedelbart efter ett misslyckat anrop till en händelseloggningsfunktion gjorts.

Tabell F-3: Exempel på GetLastError's felkoder visar några av de vanligaste felvärden som GetLastError returnerar för händelseloggningens API funktioner.

Tabell F-3: Exempel på GetLastError's felkoder

Namn	**Värde**	**Beskrivning**
ERROR_ACCESS_DENIED	5	P.g.a. otillräklig säkerhetsaccess förbjöds anropet.
ERROR_INVALID_HANDLE	6	Den HANDLE som skickades med accepterades inte som en giltig sådan.
ERROR_INVALID_PARAMETER	87	Ett icke tillåtet parametervärde försökte skickas till funktionen.

Eftersom man arbetar en hel del direkt mot disken kan även en del I/O-fel förekomma, se Tabell F-4: Vanliga I/O-fel.

Tabell F-4: Vanliga I/O-fel

Namn	**Värde**	**Beskrivning**
ERROR_FILE_NOT_FOUND	2	Den angivna backupfilen verkar inte existera.
ERROR_PATH_NOT_FOUND	3	Den angivna sökvägen till backupfilen verkar inte existera.
ERROR_HANDLE_EOF	38	Försök gjordes att läsa från en tom händelselogg.
ERROR_CANNOT_MAKE	82	Backupfilen kunde inte skapas.
ERROR_INSUFFICIENT_BUFFER	122	Läsbuffern är inte stor nog för att kunna hålla den aktuella händelsestrukturen.
ERROR_ALREADY_EXISTS	183	Den specifierade backupfilen finns redan.

Det finns också fyra felkoder som är associerade explicit med händelseloggnings APIet, se Tabell F-5: Vanliga API-felkoder

Tabell F-5: Vanliga API-felkoder

Namn	Värde	Beskrivning
ERROR_EVENTLOG_FILE_CORRUPT	1500	Händelseloggen är skadad och kan varken läsas eller skrivas till.
ERROR_EVENTLOG_CANT_START	1501	Händelseloggningsmekanismen har försökt starta med misslyckats. Detta felet returneras endast efter ett misslyckat anrop till StartService.
ERROR_LOG_FILE_FULL	1502	Händelseloggen är full och kan inte skrivas till innan den har blivit rensad.
ERROR_EVENTLOG_FILE_CHANGED	1503	Händelseloggen förändrades av en annan process under anropet.

Händelseloggaren är kapabel att läsa, skriva och göra backup av händelseloggar över nätverket på Windows NT-system. Detta görs med hjälp av ett RPC-protokoll (Remote Procedure Calls). Alla händelseloggningsfunktioner som misslyckas under ett RPC-anrop kommer att returnera ett fel till GetLastError. Tabell F-6: Vanliga RPC-felkoder visar några av dem.

Tabell F-6: Vanliga RPC-felkoder

Namn	Värde	Beskrivning
RPS_S_INVALID_NET_ADDR	1707	UNC-strängen som skickades med till funktionen är ej korrekt utformad
RPS_S_SERVER_UNAVALIABLE	1722	Fjärrtjänsten är inte tillgänglig
RPS_S_SERVER_TOO_BUSY	1723	Fjärrtjänsten är för upptagen för att kunna ta emot anropet

När man försöker fjärransluta till ett NT-system kan man även råka ut för många vanliga nätverksspecifika fel, se Tabell F-7: Felkoder i nätverksmiljö.

Tabell F-7: Felkoder i nätverksmiljö

Namn	Värde	Beskrivning
ERROR_ACCESS_DENIED	5	Processen har inte rättigheter att komma åt den delade resursen.
ERROR_NETNAME_DELETED	64	Den tidigare utdelade resursen är inte längre tillgänglig.

ERROR_BAD_NET_NAME	67	UNC strängen specifierar en sökväg som inte existerar.

UNC

Windows använder sig av en ”Universal Naming Convention” (UNC) för att identifiera system, skrivare, diskenheter, filer och namngivna pipes över ett nätverk.

En UNC är en referens till ett system eller nätverksresurs som inte använder en logisk enhetsbeteckning eller fysisk adress. UNC används av händelseloggningsmekanismens API-funktioner för att komma åt händelseloggar vid fjärranrop till NT-system.

Det finns en klar skillnad mellan UNC och den klassiska MS-DOS namngivningen LFN (Long File Name).

Ett LFN specifierar volym, sökväg, namn och filtyp och använder max 260 tecken för detta.

<enhetsbeteckning>:\<sökväg>\<filnamn>.<filtyp><NULL><NULL>

Ett exempel på ett LFN är:

C:\WINNT\CONFIG\MSADLIB.IDF

UNC använder sig av ett linkande format, men brukar varken enhetsbeteckning eller filtyp

\\<server>\<share>\<folder>\<file>

Ett exempel på UNC är:

\\MIN_DATOR\WINNT\CONFIG\MSADLIB